

P-2688



TASK SCHEDULING IN GRID ENVIRONMENT

By

P. DEVAKI

Reg. No. : 71206805002

of

KUMARAGURU COLLEGE OF TECHNOLOGY

COIMBATORE - 641 006

A PROJECT REPORT

Submitted to the

FACULTY OF INFORMATION AND COMMUNICATION ENGINEERING

In partial fulfillment of the requirements

for the award of the degree

of

MASTER OF ENGINEERING

IN

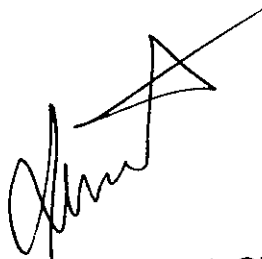
COMPUTER SCIENCE AND ENGINEERING

MAY 2009



BONAFIDE CERTIFICATE

Certified that this project report titled "Task Scheduling in Grid Environment" is the bonafide work of **Ms. P. Devaki (71206805002)** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report of dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.



SIGNATURE OF THE GUIDE

Mr. M. Nageswara Guptha M.E.,

Senior Lecturer,

Department of Computer

Science and Engineering



HEAD OF THE DEPARTMENT


Dr.S.Thangasamy Ph.D.,

Professor and Dean,

Department of Computer

Science and Engineering

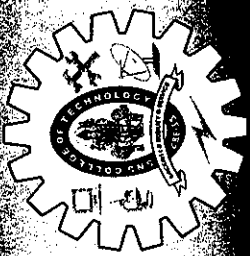
The candidate with **University Register No. 71206805002** was examined by us in Project Viva-Voce examination held on 06.07.2009



Internal Examiner



External Examiner



SNS COLLEGE OF TECHNOLOGY

Approved by AICTE and Affiliated to Anna University
(Autonomous Institution)
(Am ISO 9001:2000 Certified Institution)



Department of Information Technology
&
Society for Information Sciences and Computing Technology
Third National Conference on

NETWORKS, INTELLIGENCE AND COMPUTING SYSTEMS

Dr. Devaki M.E. CSE

This is to certify that Dr./Prof./Mr/Ms. _____ has Presented a technical paper

_____ in the National Conference on NETWORKS, INTELLIGENCE AND COMPUTING SYSTEMS held on 16th February 2009

Principal

Director cum Secretary

ACKNOWLEDGEMENT

I express my sincere thanks to our Chairman **Padmabhushan Arutselvar Dr. N.Mahalingam B.Sc, F.I.E** and Correspondent **Shri. Balasubramanian** for all their support

I would like to begin by thanking to **Dr. Joseph V. Thanikal, Ph.D.**, Principal and **Prof. R. Annamalai**, Vice-Principal for providing the necessary facilities to complete my project work.

I express my deep sense of gratitude to **Dr. S. Thangasamy, Ph.D.**, Professor and Dean, Head of the department of Computer Science and Engineering for his valuable suggestions, support and encouragement throughout the project.

I tender my special thanks to **Ms. V. Vanitha, M.E. (Ph.D.)**, Assistant Professor and Project Coordinator, Department of Computer Science and Engineering who support to complete the project successfully.

I express my heartiest thanks to my Project Guide **Mr M. Nageswara Guptha M.E. (Ph.D.)**, Senior Lecturer, Department of Computer Science and Engineering, who rendered his valuable guidance and support to perform my project work extremely well.

I also thank the teaching and non-teaching staff of our Department for providing the technical support in the duration of my project.

I also thank my friends who have supported me and helped me to complete the project work.

ABSTRACT

TASK SCHEDULING IN GRID ENVIRONMENT

Efficient task scheduling of computationally intensive applications is one of the most essential and difficult issues when aiming at high performance in a grid environment which is highly dynamic and heterogeneous. Grid computing is an emerging technology for enabling resource sharing and coordinated problem solving in dynamic multi-institutional virtual organizations. The resource matching problem in the grid involves assigning resources to tasks in order to satisfy task requirements and resource policies.

In this project work, algorithms for a Grid resource manager which is responsible for resource brokering and scheduling in Grids have been implemented. The broker selects computing resources based on the actual job requirements and the criteria in identifying the available resources, which minimizes the overall completion time (Makespan) of tasks.

This work involves Task Scheduling for dependent and independent tasks to the available pool of resources. Although a large number of scheduling heuristics have been presented in the literature, most of them target only homogeneous computing systems. For Dependent task scheduling, a simple heuristic algorithm for efficient **Heterogeneous Task Scheduling (HTS)** has been presented. The experiments have shown that HTS provides comparable or even better results for consistent, inconsistent and partially consistent environments.

For Independent task scheduling, **QoS Based Heterogeneous Task Scheduling (QBHTS)** which aims to satisfy QoS requirements of tasks has been implemented. Task is scheduled on a resource only when the resource satisfies the task's requirements. Experiment results show that the Makespan does not increase much if scheduling is performed based on QoS satisfaction.

ஆய்வுச் சுருக்கம்

அதீத செறிவு மிகு கணக்கிடுதல் தேவைப்படும் பயன்பாடுகளில் திறன்மிகு வேலைப் பங்கிடுதல் என்பது அதிக பயனுறு தன்மையைக் கருத்தில் கொண்டு வலைச் சூலலுக்கு ஏற்றவாறு வடிவமைப்பது மிகவும் முக்கியமான மற்றும் கடினமானதொரு பணியாகும். வலைச் சூழலானது ஆற்றல் மிகுந்த மற்றும் மாறுபடும் தன்மையுடைய ஒரு அமைப்பாகும். வலை என்பது ஒரு வளர்ந்து வரும் ஒரு கூட்டமைப்பாகும். மூலங்களைப் பங்கிடுதல் மற்றும் நிறுவனங்களின் செயல்பாடு ஒருங்கிணைப்பு போன்ற செயல்பாடுகளில் வலை முக்கிய பங்களிக்கின்றது.

மூலங்களை சமன்செய்யும் கணக்கீடுகளில் வலையானது மூலங்களை தகுந்த பணிகளுக்கு அளிக்கும் வேலையில் பணிகளின் தேவைகள் மற்றும் மூலங்களின் கோட்பாடுகள் ஆகியவற்றை கருத்தில் கொண்டு செயல்படுகிறது.

இந்தப் பணி வலை மூலங்களை கையாளுதலுக்கான செயல் முறைப்பற்றிய விளக்கத்தை அளிப்பதற்காக கொடுக்கப்பட்டுள்ளது. இது மூலங்களை கையாளுதல் மற்றும் பங்கிடுதல் போன்ற பணிகளை செய்கின்றது. இந்தப் பங்கீடு முறையானது தற்போதைய பணித் தேவைகள் மற்றும் மூலங்களின் இருப்பு இவற்றை கருத்தில் கொண்டு குறுகிய காலத்தில் அனைத்துப் பணிகளையும் முடிக்கின்றது. இந்த முறைமையானது மூலங்களின் இருப்பை பொருத்து சார்பற்ற மற்றும் தணித்த வேலைகளின் வேலைப்பங்கீடுகளைச் செய்கின்றது. மேலும் பல்வேறு வேலை பிரித்தளிக்கும் முறைகளைப் பற்றிய விளக்கவுரைகளும் கொடுக்கப்பட்டுள்ளன. இவற்றில் பெரும்பாலானவை ஒரே மாதிரியான சூழலில் கணக்கிடுதலையே கருத்தில் கொண்டனவாக இருக்கின்றன. ஆனால் கொடுக்கப்பட்டுள்ள முறையை சார்பற்ற வேலைகளின் பிரித்தானுதல் பற்றிய திறன்மிகு செயல்முறை விளக்கத்தை செறிவுமிகு கணக்கிடுதல் சூழலுக்கு ஏற்றவாறு வழங்குகிறது.

இம்முறை பற்றிய பகுப்பாய்வுகளும், செய்முறைகளும், இதனை கட்டமைப்பான, கட்டமைப்பற்ற மற்றும் பகுதி கட்டமைப்பான சூழல்களிலும் இதன் செயல்திறனை உறுதி செய்கின்றன.

சார்பற்ற வேலைப் பங்கிடுதலில் QoS சார்ந்த வேறுபட்ட வேலைப் பங்கீடுதளும் ஆய்வு செய்யப்பட்டுள்ளது. வேலைப் பங்கீடானது மூலங்கள் வேலைகளின் தேவைகளை நிறைவு செய்யும் பொருட்டு அவற்றிற்கு கொடுக்கப்படுகின்றன.

TABLE OF CONTENTS

| | PAGE NO. |
|---|----------|
| CONTENTS | iv |
| ABSTRACT (ENGLISH) | v |
| ABSTRACT (TAMIL) | viii |
| LIST OF FIGURES | ix |
| LIST OF TABLES | x |
| 1. INTRODUCTION | 01 |
| 1.1 PROJECT OUTLINE | |
| 1.2 PROBLEM DEFINITION | 03 |
| 1.2.1 Problem Definition for Dependent Task Scheduling | 03 |
| 1.2.2 Problem Definition for Independent Task Scheduling | |
| 2. LITERATURE SURVEY | |
| 2.1 OVERVIEW OF GRID COMPUTING | 04 |
| 2.1.1 Introduction | 05 |
| 2.1.2 Grid Architecture | 07 |
| 2.1.3 Grid Construction | 07 |
| 2.1.4 Key benefits of the Grid Computing Model | 08 |
| 2.1.5. Job Scheduling in Grids | 08 |
| 2.1.6. Issues in Grid Computing | 08 |
| 2.1.7. Grid Management | 09 |
| 2.1.8 Limitations of Grid Computing | |
| 2.2 JOB SCHEDULING | 09 |
| 2.2.1 Job Scheduling in Grid Computers | 10 |
| 2.2.2 Classification of Static Task-Scheduling algorithms | 10 |
| 2.2.3. Job Scheduling in a Heterogeneous Grid Environment | 11 |
| 2.2.4 Job Scheduling Policy for High Throughput | |

| | |
|---|----|
| 3. DETAILS OF METHODOLOGIES | 12 |
| 3.1 Application Representation for dependent jobs | 13 |
| 3.2 Existing Algorithms taken up for comparison | 13 |
| 3.3 Drawbacks of existing algorithms | 14 |
| 3.4 Proposed method - Heterogeneous Task Scheduling (HTS) | 15 |
| 3.5. Application Representation for Independent jobs | 16 |
| 3.6 Existing Algorithms taken up for comparison | 16 |
| 3.7 Drawbacks of existing algorithms | 16 |
| 3.8 Proposed method - QoS Based Heterogeneous Task Scheduling (QBHTS) | 18 |
| 3.9 Implementation Detail | 19 |
| 4. EXPERIMENTAL RESULTS | |
| 4.1 Experimental Results and Discussions for Dependent jobs | 32 |
| 4.2 Experimental Results and Discussions for Independent jobs | 38 |
| 5. CONCLUSION AND FUTURE ENHANCEMENTS | 39 |
| APPENDIX | 56 |
| REFERENCES | |

LIST OF FIGURES

| FIGURE NO. | NAME | PAGE NO. |
|---------------|---|----------|
| 2.1 | A layered grid architecture and its relationship to the Internet protocol architecture. | 5 |
| 2.2 | Classification of Static task-Scheduling algorithms | 10 |
| 3.1 | Task Graph Representation | 12 |
| 3.2 | Makespan Comparison | 14 |
| 4.1 to 4.12 | Makespan Comparison – Dependent Jobs | 20 |
| 4.13 to 4.24 | Speedup Comparison - Dependent jobs | 24 |
| 4.25 to 4. 36 | Makespan Comparison – Independent Jobs | 32 |

LIST OF TABLES

| TABLE NO. | NAME | PAGE NO. |
|------------------|---|-----------------|
| 3.1 | Weight matrix representation | 13 |
| 4.1 to 4.12 | Number of Favorable Cases for 1000 Trials | 28 |

LIST OF ABBREVIATIONS

| ABBREVIATION | EXPANSION |
|--------------|--|
| QoS | Quality of Service |
| DAG | Directed Acyclic Graph |
| QBHTS | QoS Based Task Scheduling |
| OGSA | Open Grid Services Architecture |
| API | Application Provider Interface |
| COTS | Commercial off-the-shelf |
| GRAM | Grid Resource Allocation Manager |
| MDS | Monitoring and Directory Service |
| GRIS | Grid Resource Information Service |
| GIIS | Global Index Information Service |
| GASS | Global Access to Secondary Storage |
| LDAP | Light-Weighted Directory Access Protocol |
| CPOP | Critical Path on a Processor |
| HCPT | Heterogeneous Critical Parent Trees |
| HTS | Heterogeneous Task Scheduling |
| ETC | Expected Time to Compute |
| CCR | Communication to Computation Ratio |

CHAPTER 1

INTRODUCTION

1.1 PROJECT OUTLINE

Recent developments in high-speed digital communication have made it possible to connect a distributed suite of different high performance machines in order to provide a powerful computing platform called a *heterogeneous computing system*. This platform is utilized to execute computationally intensive applications that have diverse computation requirements. This has resulted in the ability to form loosely coupled, high-performance computational environment comprising numerous scalable, fault tolerant, and platform-independent services across the entire Internet. The grid infrastructure provides a way to execute applications over autonomous, distributed and heterogeneous nodes by secure resource sharing among individuals and institutions. Typically, a user can submit jobs to a grid without necessarily knowing (or even caring) where it will be executed. It is the responsibility of the grid resource management system to distribute such jobs among a heterogeneous pool of servers, trying to optimize the resource usage and provide the best possible quality of service.

This project deals with the applications with dependent and independent tasks. The performance of parallel applications on such systems is highly dependent on the scheduling of the application tasks onto these machines. The main objective of the *scheduling mechanism* is to map tasks onto machines and order their executions so that precedence requirements are satisfied and minimum overall completion time is achieved (Makespan). When the structure of the parallel application in terms of its task execution times, task dependencies and size of communicated data is known a priori, the application is represented with the static model, and scheduling can be accomplished statically at compile time. In the general form of static task scheduling, the application is represented by the *directed acyclic graph* (DAG), in which the nodes represent application tasks and the edges represent inter-task data dependencies. Each node is labeled by the computation cost (expected computation time) of the task and each edge is labeled by the communication cost (expected communication time). HTS aims to reduce the Makespan.

For independent tasks, the QoS Based Task Scheduling (QBHTS) provides management for quality of service on different types of resources, including networks, CPUs, and disks. It also encourages Grid customers to specify their quality of service needs based on their actual requirements. The main goal of this system is to provide seamless access to users for submitting jobs to a pool of heterogeneous resources, and at the same time, dynamically scheduling in multi policy mode and monitoring the resource requirements for execution of applications.

1.2 PROBLEM DEFINITION

1.2.1 PROBLEM DEFINITION FOR DEPENDENT TASK SCHEDULING

To determine the assignment of Tasks (N) of a given application to a given machine set P ($P < N$) such that

- The scheduling length (**Makespan** – overall completion time) is to be minimized
- All precedence constraints are to be satisfied for dependent jobs.
- The application is represented by the Task Graph
- The Resources are scheduled in Batch Mode, where the jobs and resources are collected and mapped at prescheduled time.

1.2.2 PROBLEM DEFINITION FOR INDEPENDENT TASK SCHEDULING

The resource matching problem in the Grid involves assigning P resources to N tasks where $P < N$, in order to satisfy task requirements and resource policies. The broker selects computing resources based on actual task requirements and a number of criteria identifying the available resources, with the aim to minimize the turnaround time for the individual application. The problem is to match the resources for the required tasks in grid environment. The Resources are scheduled in Batch Mode, where the jobs and resources are collected and mapped at prescheduled time.

CHAPTER 2

LITERATURE SURVEY

2.1. OVERVIEW OF GRID COMPUTING

2.1.1. INTRODUCTION

Grid computing is a form of distributed computing that involves coordinating and sharing computing, application, data, storage, or network resources across dynamic and geographically dispersed organizations [5] [6]. Grid technologies promise to change the way organizations tackle complex computational problems. Grid computing enables the virtualization of distributed computing and data resources such as processing, network bandwidth and storage capacity to create a single system image, granting users and applications seamless access to vast IT capabilities.

Grid computing is based on an open set of standards and protocols — e.g., **Open Grid Services Architecture (OGSA)** — that enable communication across heterogeneous, geographically dispersed environments.

"A Grid is a collection of distributed computing resources available over a local or wide area network that appears to an end user or application as one large virtual computing system." Another definition is "Grid computing is computing as a utility - you do not care where data resides, or what computer processes your requests. Analogous to the way utilities work, clients request information or computation and have it delivered - as much as they want, and whenever they want."

Grid computing represents an enabling technology that permits the dynamic coupling of geographically dispersed resources (machines, networks, data storage, visualization devices, software and scientific instruments) for performance-oriented distributed applications in science, engineering, medicine and e-commerce.

The first goal is to build up a **computational and networking infrastructure** that is designed to provide pervasive, uniform and reliable access to data, computational and human resources distributed over wide area environments. So a grid should bring together

a diverse collection of different hardware and software technologies, different corporations, people and procedures do build a shared pool of resources.

The second and more distant goal behind grid computing is the **delivery of computing power** as a utility, like the electrical system. Actually the name 'Grid' comes from an analogy from power grids that supply electricity. When somebody needs electricity, he plugs in a device to the system which uses as much resources as it needs. The end user is not concerned with the details like which power plant is supplying the electricity at that moment.

2.1.2 GRID ARCHITECTURE

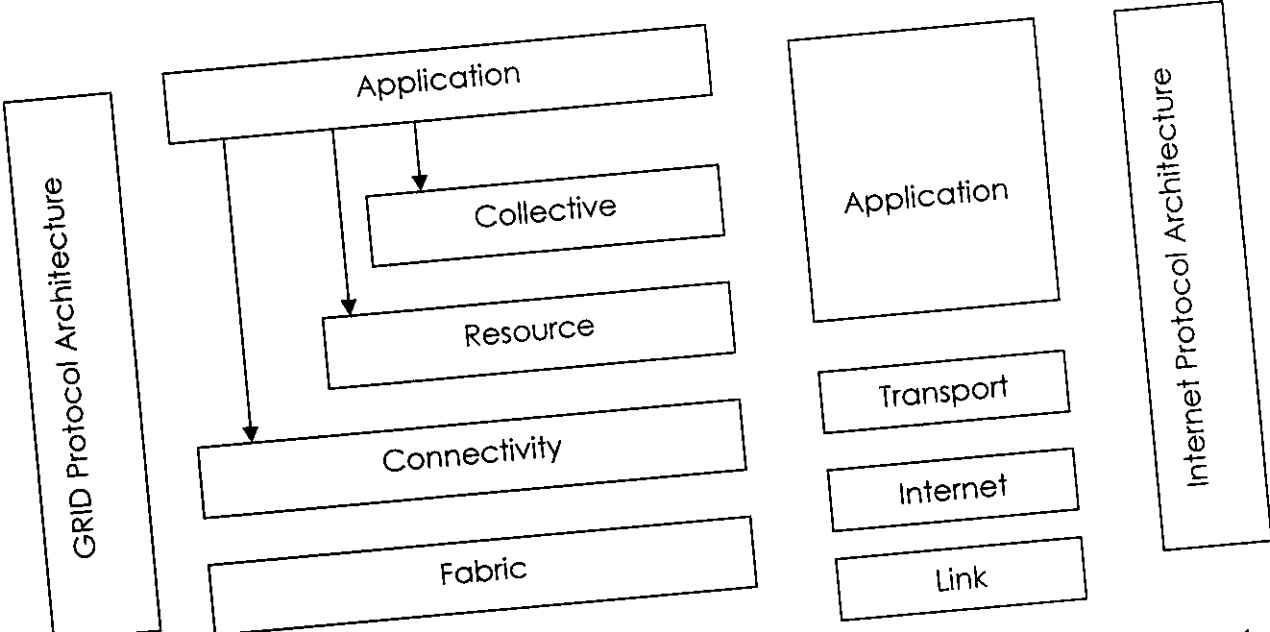


Figure 2.1 A layered grid architecture and its relationship to the Internet protocol architecture.

Figure 2.1 illustrates the component layers of the architecture with specific capabilities at each layer. Each layer shares the behavior of the component layers. Each of these component layers is compared with their corresponding Internet protocol Layers, for purposes of providing more clarity in their capabilities.

Fabric Layer: Interface to Local Resources

This defines the resources that can be shared. This could include computational resources, data storage, networks, catalogs and other system resources. These resources can be physical resources or logical resources by nature. Example for logical resources are distributed file systems, computer clusters etc.,

Basic capabilities are

1. Provide an "inquiry" mechanism whereby it allows for the discovery against its own resource capabilities, structure and state of operations.
2. Provide appropriate "resource management" capabilities to control the QoS the grid solution promises or has been contracted to deliver.

Connectivity Layer: Manages Communications

This defines the core communication and authentication protocol required for grid-specific networking services transactions. It includes networking transport, routing and naming. Characteristics to be considered are Single sign on, Delegation, User-Based trust relationships and Data Security.

Resource Layer: Sharing of a Single Resource

This utilizes the communication and security protocol defined by the networking communications layer, to control the secure negotiation, initiation, monitoring, metering, accounting, and payment involving the sharing of operations across individual resources.

The Collective Layer: Coordinating Multiple Resources

While the Resource layer manages an individual resource, the Collective layer is responsible for all global resource management and interaction with a collection of resources. Collective services are Discovery Services, Co allocation, Scheduling and Brokering Services, Monitoring and Diagnostic Services, Data Replication Services etc.,

Application Layer: User- Defined Grid Applications

These are user applications, which are constructed by utilizing the services defined at each lower layer. Such an application can directly access the resource, or can access the resource through the Collective service interface APIs (Application Provider Interface)

2.1.3 GRID CONSTRUCTION

There are three main issues that characterize computational grids:

Heterogeneity : A grid involves a multiplicity of resources that are heterogeneous in nature and might span numerous administrative domains across wide geographical distances.

Scalability: A grid might grow from few resources to millions.

Dynamicity or Adaptability: With so many resources in a Grid, the probability of some resource failing is naturally high.

2.1.4. KEY BENEFITS OF THE GRID COMPUTING MODEL

Consolidation: Consolidation is a key benefit of the Grid computing model, especially in the data center. Consolidation not only minimizes the infrastructure necessary to meet an enterprise's business demands, but also reduces costs by migrating from proprietary or single-use systems to **commercial off-the-shelf (COTS)**-based systems that can be shared by multiple applications.

Modular Computing: Modular computing, especially in the data center, minimizes and simplifies the infrastructure using building blocks that address higher density, lower power, lower thermals, simplified cabling, and ease of upgrading and management. (Blade Servers)

Virtualization : By creating pools of resources enabled by highly automated management capabilities, virtualization can enable an IT system administrator to utilize far more of the resources in the data center, making the resources accessible to more than a single application sitting on a single physical server.

Utility Computing : Utility Computing allows an infrastructure to be managed analogously to an electric utility, applying a pay-per-use model, thereby optimizing and balancing the computing needs of an enterprise, and allowing it to run at maximum efficiency.

2.1.5 Job Scheduling in Grids [4]

The job scheduling system is responsible to select best suitable machines in a grid for user jobs. The management and scheduling system generates job schedules for each machine in the grid by taking static restrictions and dynamic parameters of jobs and machines into consideration.

2.1.6. Issues in Grid Computing [11]

A grid is a **distributed and heterogeneous** environment. Being heterogeneous inherently contains the problem of managing multiple technologies and administrative domains. In Grid, **security** is a main issue. The users who submit their tasks and their data to the grid wish to make sure that their programs and data is not stolen or altered by the computer in which it is running. Another important issue is **scheduling**. Scheduling a task to the correct resource requires considerable effort. The picture is further complicated when we consider the need to access the data.

2.1.7 Grid Management [10]

One of the major problems in grid computing is to be able to schedule jobs and data to a suitable resource. As a grid may contain many different hardware and software configurations, a standard has to be agreed upon. The most widely used product for managing a grid is called **Globus Toolkit**. Supported by many large vendors, Globus offers all the functionality needed to manage a grid system.

Grid Resource Allocation Manager (GRAM) allows users to select a specific resource in the grid to run their jobs on. It has a client side module that allows user to schedule jobs at a specific server in the grid and a gatekeeper module that is running in each server to schedule arriving jobs. GRAM makes use of **Monitoring and Directory Service (MDS)**. MDS manages a directory of local and global resources. **Grid Resource Information Service (GRIS)** collects local resource information. **Global Index Information Service**

(GIIS) collects GRIS information from all servers and provides a centralized resource directory for the whole grid. The movement of data in the grid is managed by **Global Access to Secondary Storage (GASS)**.

Apart from these basic services, Globus provides security functions and packaging tools to deploy software in a format that would work in any server. The reason behind the success of Globus is the open source approach and use of standards. For example, Globus uses SSL for secure data transfer, **Light-Weighted Directory Access Protocol (LDAP)** for directory information. By using these standard protocols it ensures that it is compatible with many operation environments.

2.1.8 Limitations of Grid Computing

Not every application is suitable or enabled for running on a grid. For example some kinds of applications simply cannot be parallelized. For others, it can take a large amount of work to modify them to achieve faster throughput. The configuration of a grid can greatly affect the performance, reliability, and security of an organization's computing infrastructure.

2.2 JOB SCHEDULING

2.2.1 Job Scheduling in Grid Computers [11]

Distributed computing utilizes a network of many computers, each accomplishing a portion of an overall task, to achieve a computational result much more quickly than with a single computer. In distributed computing the task is split up into smaller chunks and performed by the many computers owned by the general public. The key issue here is that we are using computing power that we don't own. These computers are owned and controlled by other people, who you would not necessarily trust.

Grid computing is a form of distributed computing that coordinates and shares computation, application, and data storage or network resources across dynamic and geographically dispersed organizations. One primary issue associated with the efficient utilization of heterogeneous resources in a grid is grid scheduling. Grid scheduling is a challenge because the capability and availability of resources vary dynamically. The complexity of scheduling problem increases with the size of the grid and becomes difficult to solve effectively. Challenging tasks are, searching for resources in the collection of

geographically distributed heterogeneous computing systems and making scheduling decisions, taking into consideration quality of service. Grid scheduler does not have full control over the grid. The grid scheduler can not assume that it has a global view of the grid.

2.2. 2 Classification of Static Task-Scheduling algorithms [2]

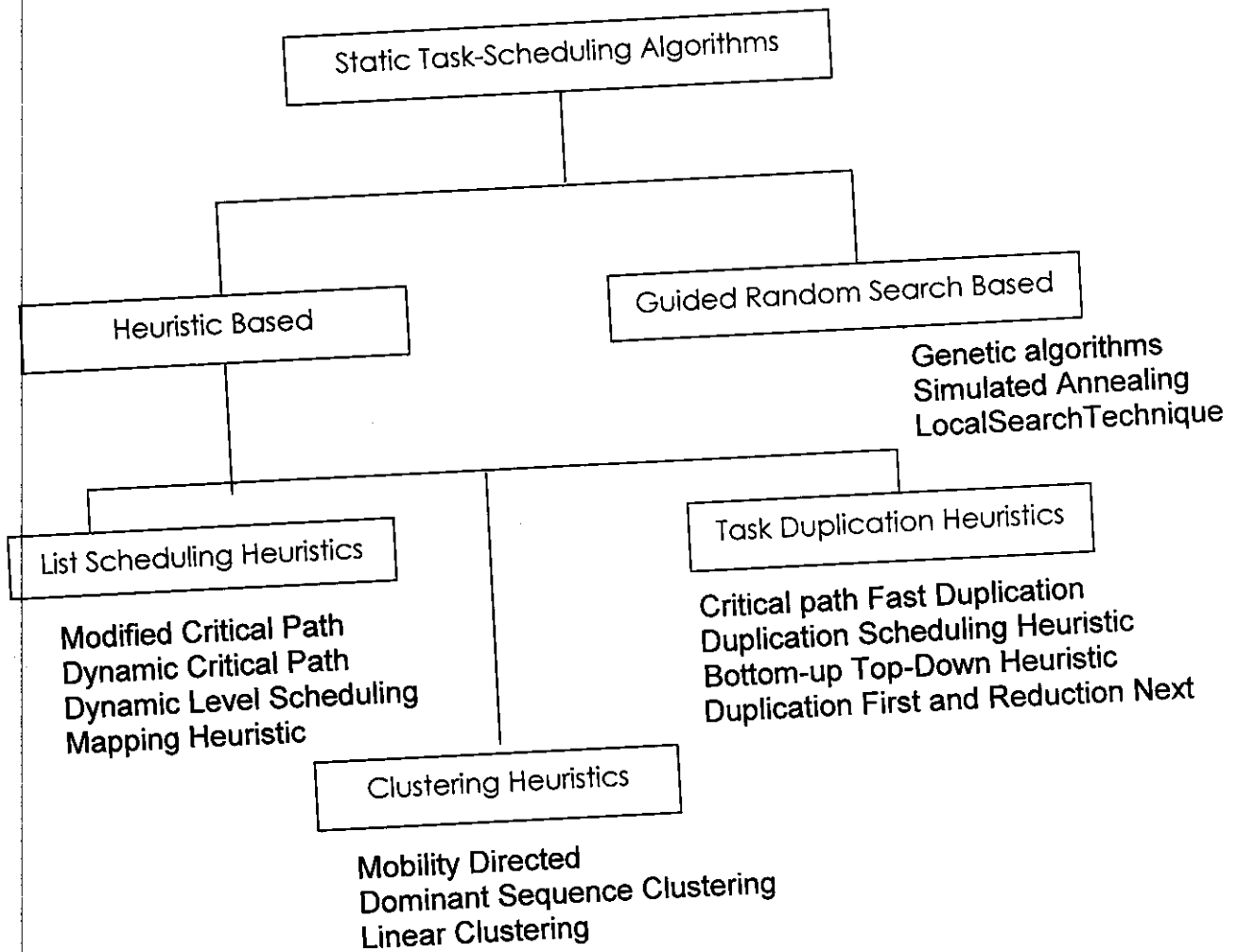


Figure 2.2 Classification of Static task-Scheduling algorithms

2.2.3. Job Scheduling in a Heterogeneous Grid Environment [10]

Computational grids have the potential for solving large-scale scientific problems using heterogeneous and geographically distributed resources. However, a number of major

technical hurdles must be overcome before this potential can be realized. One problem that is critical to effective utilization of computational grids is the efficient scheduling of jobs.

One of the primary goals of grid computing is to share access to geographically distributed heterogeneous resources in a transparent manner. There will be many benefits when this goal is realized, including the ability to execute applications whose computational requirements exceed local resources and the reduction of job turnaround time through workload balancing across multiple computing facilities. The development of computational grids and the associated middleware has therefore been actively pursued in recent years. However, many major technical (and political) hurdles stand in the way of realizing these benefits. Although numerous researchers have proposed scheduling algorithms for parallel architectures, the problem of scheduling jobs in a heterogeneous grid environment is fundamentally different.

2.2.4 Job Scheduling Policy for High Throughput

The growing computational power requirements of grand challenge applications has promoted the need for merging high throughput computing and grid Computing principles to harness computational resources distributed across multiple organizations. First of all there is a lot of activity to bring standards to the field. Globus is a big step forward towards the formation of very large global grid systems. Secondly, the hardware vendors are rushing to deliver the right kind of hardware for this new architecture. Blade servers will make it possible in the future that whenever we have a job, there will be an available server somewhere to execute it. Software vendors like Oracle are also delivering products that take advantage of these new architectures.



CHAPTER 3

DETAILS OF METHODOLOGIES

3.1 APPLICATION REPRESENTATION FOR DEPENDENT JOBS

3.1.1 Task Graph Representation

- Directed Acyclic Graph: $G(V,E)$
- V is the set of v nodes, each node $v_i \in V$ represents an application Task, which is a sequence of instructions that must be executed serially on the same machine.
- E is the set of communication edges. The directed edge $e_{i,j}$ joins nodes v_i and v_j , where node v_i is called the parent node and node v_j is called child node. This also implies that v_j cannot start until v_i finishes and sends its data to v_j .
- $C_{i,j}$ is the communication cost from the node n_i to the node n_j .

3.1.2 Application Representation using Task Graph

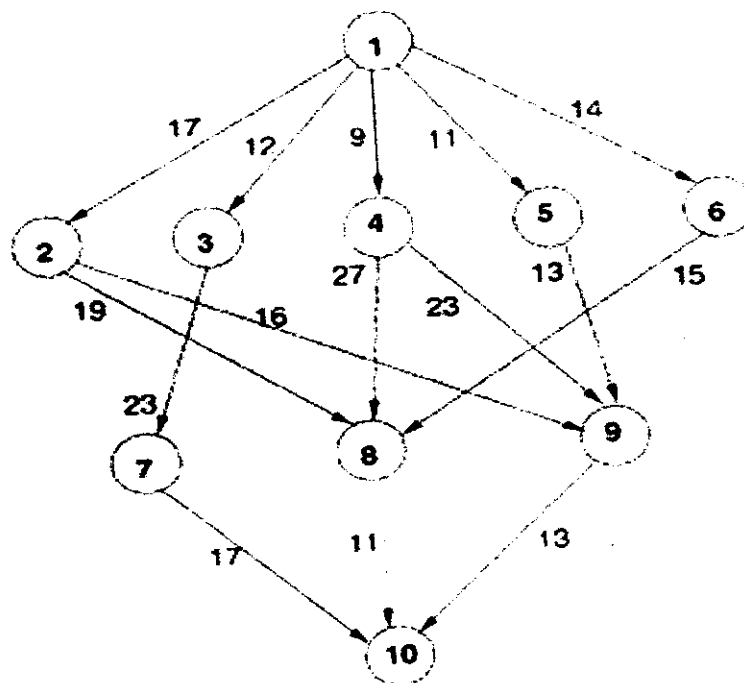


Figure 3.1 Task graph representation

3.1.3 Representation of Weight Matrix

| TASK | P1 | P2 | P3 |
|------|----|----|----|
| 1 | 14 | 16 | 9 |
| 2 | 13 | 19 | 18 |
| 3 | 11 | 13 | 19 |
| 4 | 13 | 8 | 17 |
| 5 | 12 | 13 | 10 |
| 6 | 13 | 16 | 9 |
| 7 | 7 | 15 | 11 |
| 8 | 5 | 11 | 14 |
| 9 | 18 | 12 | 20 |
| 10 | 21 | 7 | 16 |

Table 3.1 Weight matrix representation

3.2. EXISTING ALGORITHMS TAKEN UP FOR COMPARISON

Heterogeneous Critical Parent Trees (HCPT) [1] and Critical Path on a Processor (CPOP) [2] algorithms are taken up for the comparison. The weight matrix has been generated using simulation model [3].

3.3 DRAWBACKS OF EXISTING ALGORITHMS

Existing algorithms use almost 50% of its total execution time for computing listing phase [1] [2]. In order to avoid this calculation, a ready Queue is dynamically maintained. After executing a particular task, ready queue is updated with its children if they become ready tasks.

The algorithm starts from the entry node. Initially the entry node is available in ready queue. Until the queue is empty take the task of the nodes in ready queue. Select the node (let task t_i) which has earliest completion time at machine m_j and remove from the ready queue. Allocate the task t_i in m_j . Update the ready queue by adding the tasks which are ready due to t_i completion. The overall completion time (makespan) is calculated.

3.4. PROPOSED METHOD - HETEROGENEOUS TASK SCHEDULING (HTS)

In this method dynamic Ready Queue (RQ) is calculated after scheduling each task.

Algorithm

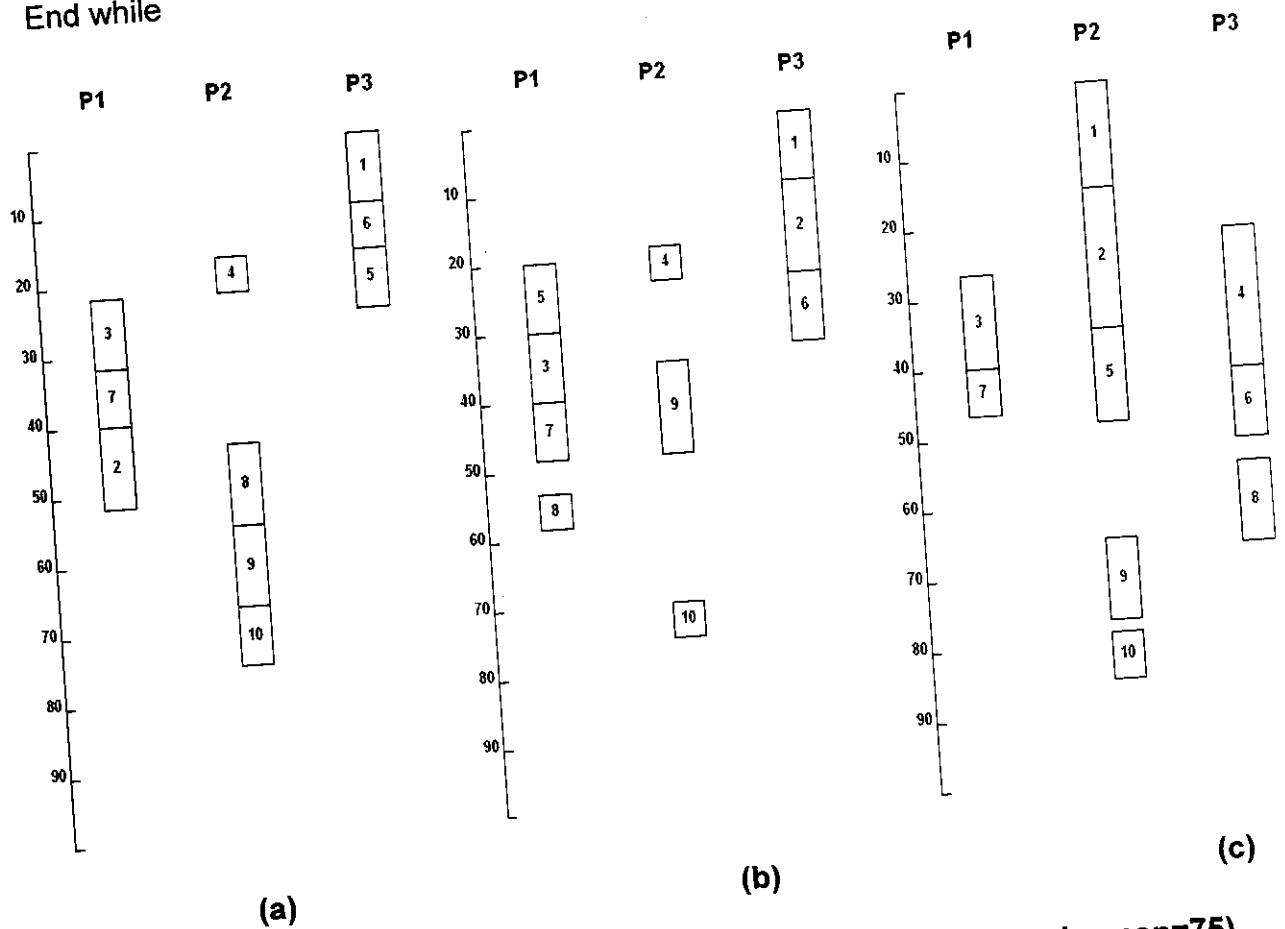
Initialize the Ready queue (RQ) with the entry task

While there is an unscheduled task in RQ do

Assign the task n_i to processor p_j which minimizes the EEFT (n_i, p_j)

Update RQ with the successors of n_i , if they become ready tasks

End while



Scheduling of the task graph in the figure with: (a) HTS(makespan=75)
 (b)HCPT(makespan=76) (c) CPOP(makespan=86)

Figure 3.2 Makespan comparison

3.5 APPLICATION REPRESENTATION FOR INDEPENDENT JOBS

Following model explains the problem:

1. The problem input
 - a. A set of resources with their capabilities
 - b. A set of tasks with their requirements
2. The problem output: Matching the best resource for each task
3. The problem purpose: Minimizing turnaround time

The following parameters are considered:

- n : the task number
- m : the resource number
- k : the number of QoS parameters
- q^{res} : Resource Capability
- q^{task} : task requirement

The vector q^{res} which gives the capabilities of a resource is as follows:

$$q^{res} = \langle q_1^{res}, q_2^{res}, \dots, q_k^{res} \rangle \quad (1)$$

The requirements of a resource are given by the vector with QoS parameters and weights for the parameters are given in the following equations.

$$q^{task} = \langle q_1^{task}, q_2^{task}, \dots, q_k^{task} \rangle \quad (2)$$

$$W = \langle w_1, w_2, \dots, w_k \rangle \quad 0 \leq w_i \leq 1 \quad \sum_{i=1}^k w_i = 1 \quad (3)$$

The satisfy operator \bowtie is introduced. $R_i \bowtie T_j$ means that the resource R_i can satisfy the task T_j and guarantees QoS parameters.

$$R_i \bowtie T_j = \left(\left(\sum_{l=1}^k \frac{q_l^{Res_i}}{q_l^{task_j}} \times w_l \right) \geq 1 \right)$$

(k = the number of QoS parameters) (4)

3.6 EXISTING ALGORITHM TAKEN UP FOR COMPARISON

The existing algorithm Min-Min Algorithm [8] has been taken up for comparison in which Task Requirement and its satisfaction are not considered.

3.7. DRAWBACKS OF EXISTING ALGORITHMS

The existing independent job scheduling algorithms are mostly concentrating only on reducing the makespan but they are not concentration on satisfying the requirements of task. In this method the requirements are unit less given by the tasks. Machines are allotted to these tasks only when the machine satisfies the requirements.

3.8 PROPOSED METHOD - QOS BASED HETEROGENEOUS TASK SCHEDULING (QBHTS)

There are three matrices, one is $T_{n \times k}$ matrix given by (5) for task requirements, another is $W_{n \times k}$ matrix given by (6) for weight of requirements, and the other is $R_{k \times m}$ matrix given by (7) for resource capabilities.

These matrices are shown in below.

$$T_{n \times k} = \begin{bmatrix} q_1^{task_1} & q_2^{task_1} & \dots & q_k^{task_1} \\ q_1^{task_2} & q_2^{task_2} & \dots & q_k^{task_2} \\ \vdots & \vdots & \ddots & \vdots \\ q_1^{task_n} & q_2^{task_n} & \dots & q_k^{task_n} \end{bmatrix} \quad (5)$$

$$W_{n \times k} = \begin{bmatrix} w_1^{task_1} & w_2^{task_1} & \dots & w_k^{task_1} \\ w_1^{task_2} & w_2^{task_2} & \dots & w_k^{task_2} \\ \vdots & \vdots & \ddots & \vdots \\ w_1^{task_n} & w_2^{task_n} & \dots & w_k^{task_n} \end{bmatrix} \quad (6)$$

$$R_{k \times m} = \begin{bmatrix} q_1^{res_1} & q_1^{res_2} & \dots & q_1^{res_m} \\ q_2^{res_1} & q_2^{res_2} & \dots & q_2^{res_m} \\ \vdots & \vdots & \ddots & \vdots \\ q_k^{res_1} & q_k^{res_2} & \dots & q_k^{res_m} \end{bmatrix} \quad (7)$$

Defined the matrix $WdT_{n \times k}$ as below:

$$WdT_{n \times k} = \begin{bmatrix} W_1^{task_1} / q_1^{task_1} & W_2^{task_1} / q_2^{task_1} & \dots & W_k^{task_1} / q_k^{task_1} \\ W_1^{task_2} / q_1^{task_2} & W_2^{task_2} / q_2^{task_2} & \dots & W_k^{task_2} / q_k^{task_2} \\ \vdots & \vdots & \ddots & \vdots \\ W_1^{task_n} / q_1^{task_n} & W_2^{task_n} / q_2^{task_n} & \dots & W_k^{task_n} / q_k^{task_n} \end{bmatrix} \quad (8)$$

So, equation (4) is based on multiplying $WdT_{n \times k}$ matrix to $R_{k \times m}$ matrix and the result is $V_{n \times m}$ matrix given by (9) and (10).

$$V_{n \times m} = WdT_{n \times k} * R_{k \times m} \quad (9)$$

$$V_{n \times m} = \begin{bmatrix} \sum_{i=1}^k \left(\frac{W_i^{task_1}}{q_i^{task_1}} * q_i^{res_1} \right) & \sum_{i=1}^k \left(\frac{W_i^{task_1}}{q_i^{task_1}} * q_i^{res_2} \right) & \dots & \sum_{i=1}^k \left(\frac{W_i^{task_1}}{q_i^{task_1}} * q_i^{res_m} \right) \\ \sum_{i=1}^k \left(\frac{W_i^{task_2}}{q_i^{task_2}} * q_i^{res_1} \right) & \sum_{i=1}^k \left(\frac{W_i^{task_2}}{q_i^{task_2}} * q_i^{res_2} \right) & \dots & \sum_{i=1}^k \left(\frac{W_i^{task_2}}{q_i^{task_2}} * q_i^{res_m} \right) \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^k \left(\frac{W_i^{task_n}}{q_i^{task_n}} * q_i^{res_1} \right) & \sum_{i=1}^k \left(\frac{W_i^{task_n}}{q_i^{task_n}} * q_i^{res_2} \right) & \dots & \sum_{i=1}^k \left(\frac{W_i^{task_n}}{q_i^{task_n}} * q_i^{res_m} \right) \end{bmatrix} \quad (10)$$

V_{ij} shows the value of (4) for assigning resource j to task i . If $V_{ij} = 1$, the resource j exactly will provide the task i requirements. If $V_{ij} < 1$, the resource j will be weaker than task i requirements. If $V_{ij} > 1$, the resource j will be stronger than task i requirements.

Algorithm

- The $V_n \times m$ matrix is generated.

Depending on the value of $V_n \times m$ matrix resource matching is done as follows:

The Max $M_{n \times 3}$ matrix is generated using the following steps

- In $V_n \times m$ matrix first 16 tasks are taken and repeat the following for 512 tasks.
- Maximum satisfaction factor is selected for the first row and allocated to the respective machine.
- Then the maximum satisfaction factor in the next row is selected and checked whether the corresponding machine is already allocated or not.
- If allocated then the next maximum satisfaction is selected from the same row and check for the availability of the respective machine. If not available, proceed with the next maximum until the task is assigned to the idle machine.
- This process is repeated until all the machines are allocated to some tasks.

3.9 IMPLEMENTATION DETAIL

3.9.1 Input Weight Matrix Generation

Input Weight Matrix is generated using the simulation model [3]. This is also known as **ETC – Expected Time to Compute Matrix**.

3.9.2 Graph Construction

The random graph generator was implemented to generate application graphs with various characteristics. The generator requires the following input parameters:

- number of tasks in the graph v ,
- The computation cost w_i for each task t_i is generated using the simulation model.
- **Communication to Computation Ratio (CCR)**, which is defined as the ratio of the average communication cost to the average computation cost.
- Each node in the level l_i has half the number of nodes in the level l_{i-1} as parents.

In all experiments

- Only graphs with a single entry and a single exit node were considered.
- Graph levels $l=5$.

CHAPTER 4

EXPERIMENTAL RESULTS

4.1 Experimental Results and Discussions for Dependent jobs

This section presents performances comparison of the proposed algorithm with the existing CPOP and HCPT algorithms.

4.1.1 Comparison Metrics

The comparisons of the algorithms are based on the following metrics:

4.1.1.1 Makespan

The makespan, or scheduling length, is defined as:

$$\text{Makespan} = FT(v_{\text{exit}}) ,$$

Where $FT(v_{\text{exit}})$ is the finishing time of the scheduled exit node.

4.1.1.2 Speedup

The speedup value is defined as the ratio of the sequential execution time (i.e., cumulative computation costs of all tasks) to the parallel execution time (i.e., the makespan). The sequential execution time is computed by assigning all tasks to a single machine, which minimizes the cumulation of the computation costs.

$$\text{SpeedUp} = (\min_{p_j \in Q} \{\sum_{n_i \in V} w_{i,j}\}) / \text{makespan}$$

4.1.2 Comparison Graphs

4.1.2.1 Makespan Comparison – Dependent Jobs

Low Task Heterogeneity Low Machine Heterogeneity

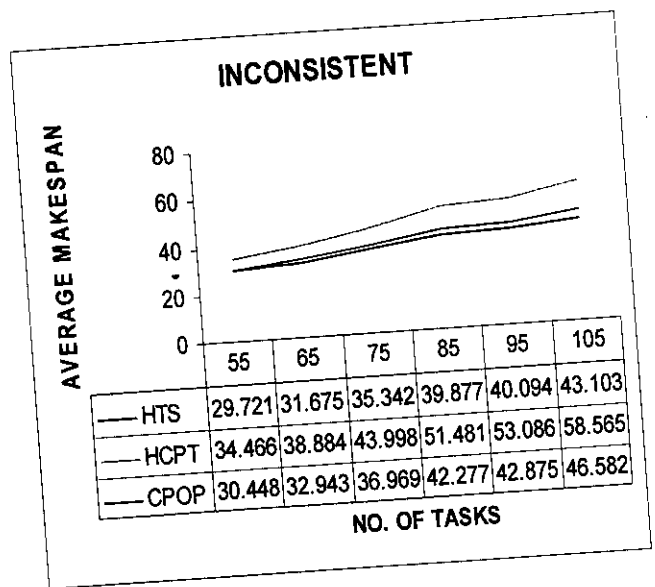


Figure 4.1

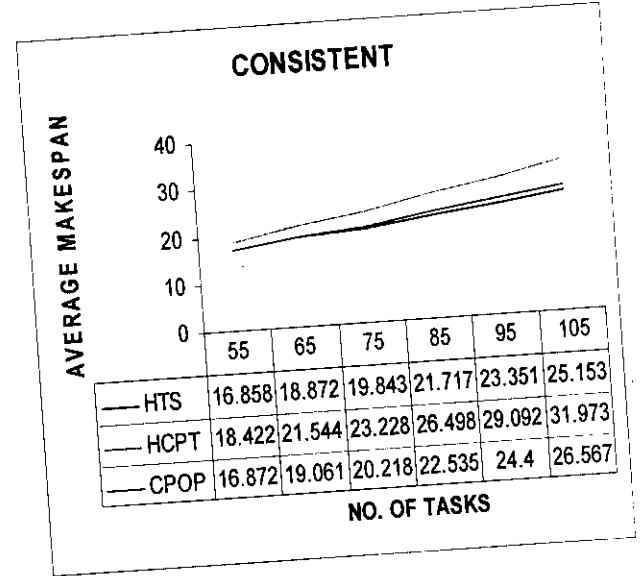


Figure 4.2

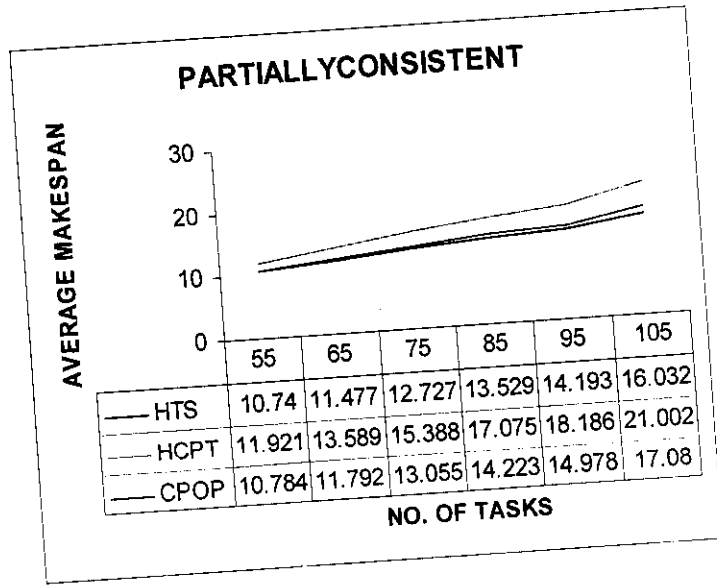


Figure 4.3

Low Task Heterogeneity High Machine Heterogeneity

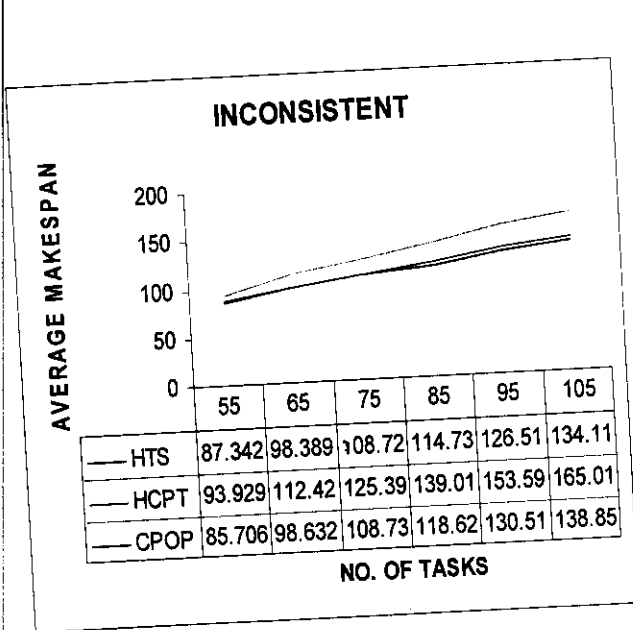


Figure 4.4

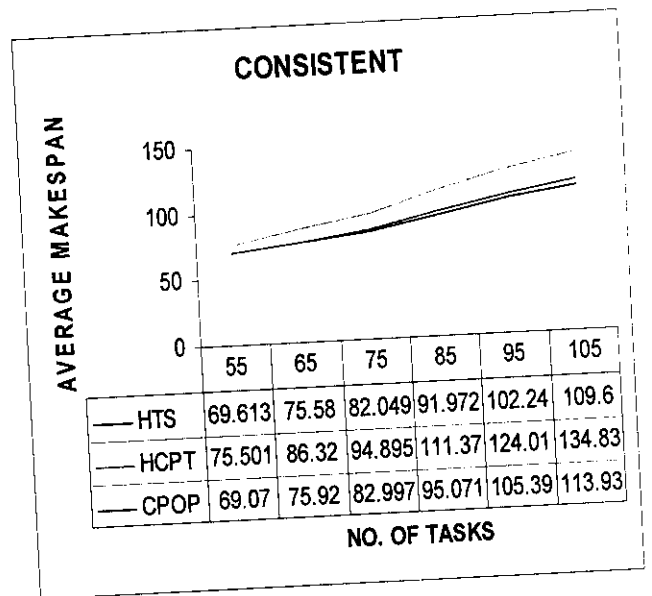


Figure 4.5

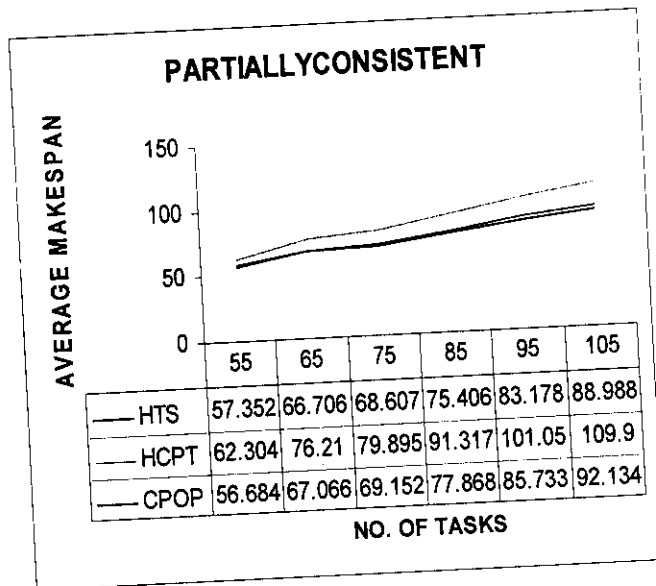


Figure 4.6

High Task Heterogeneity Low Machine Heterogeneity

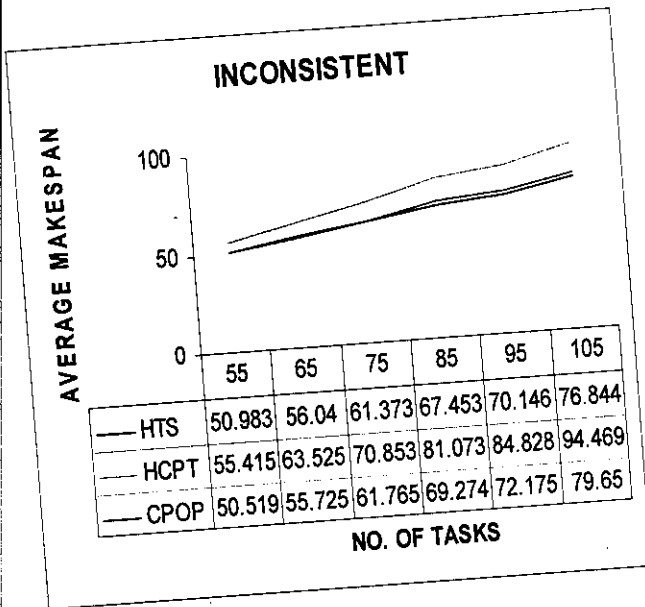


Figure 4.7

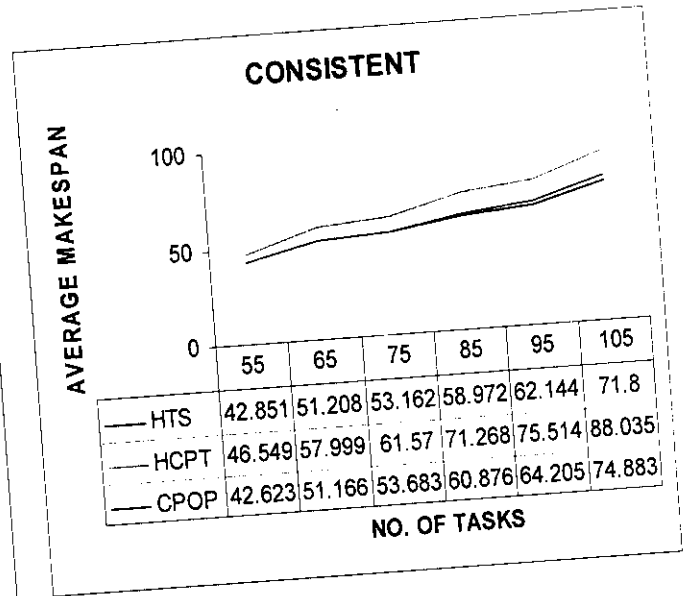


Figure 4.8

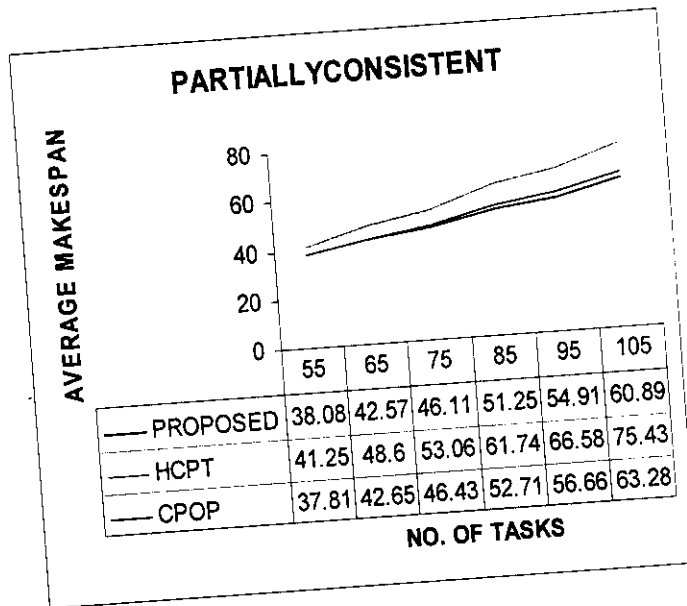


Figure 4.9

High Task Heterogeneity High Machine Heterogeneity

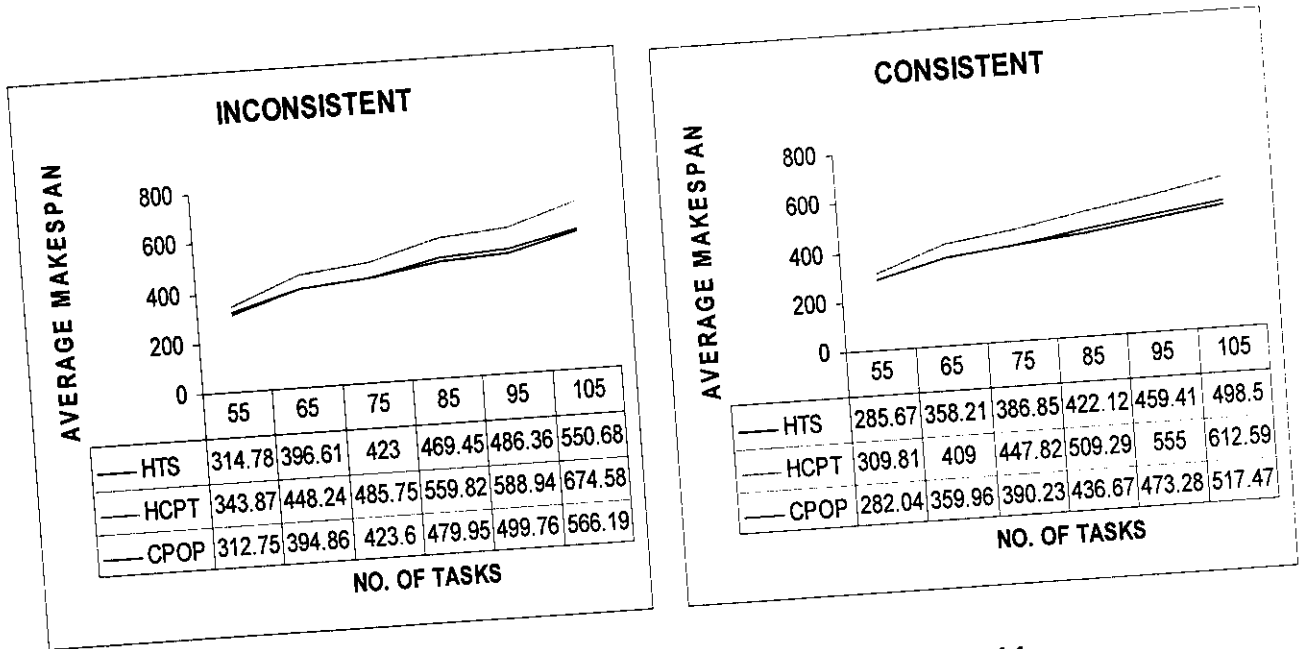


Figure 4.10

Figure 4.11

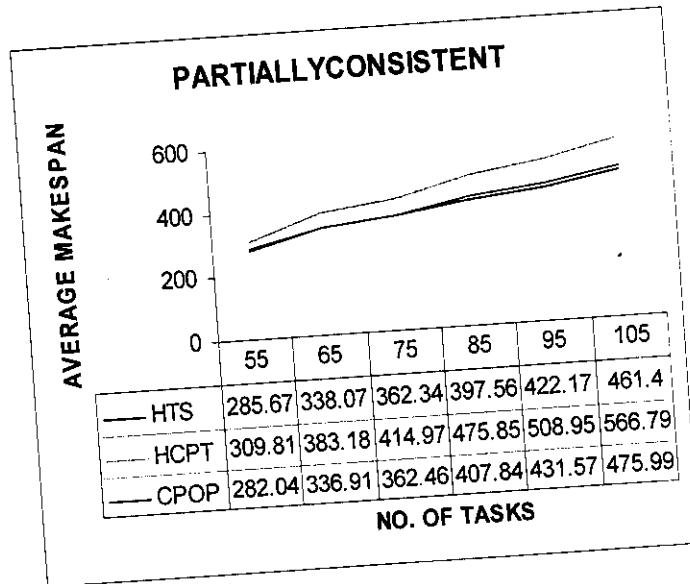


Figure 4.12

4.1.2.2 Speedup Comparison – Dependent Jobs

Low Task Heterogeneity Low Machine Heterogeneity

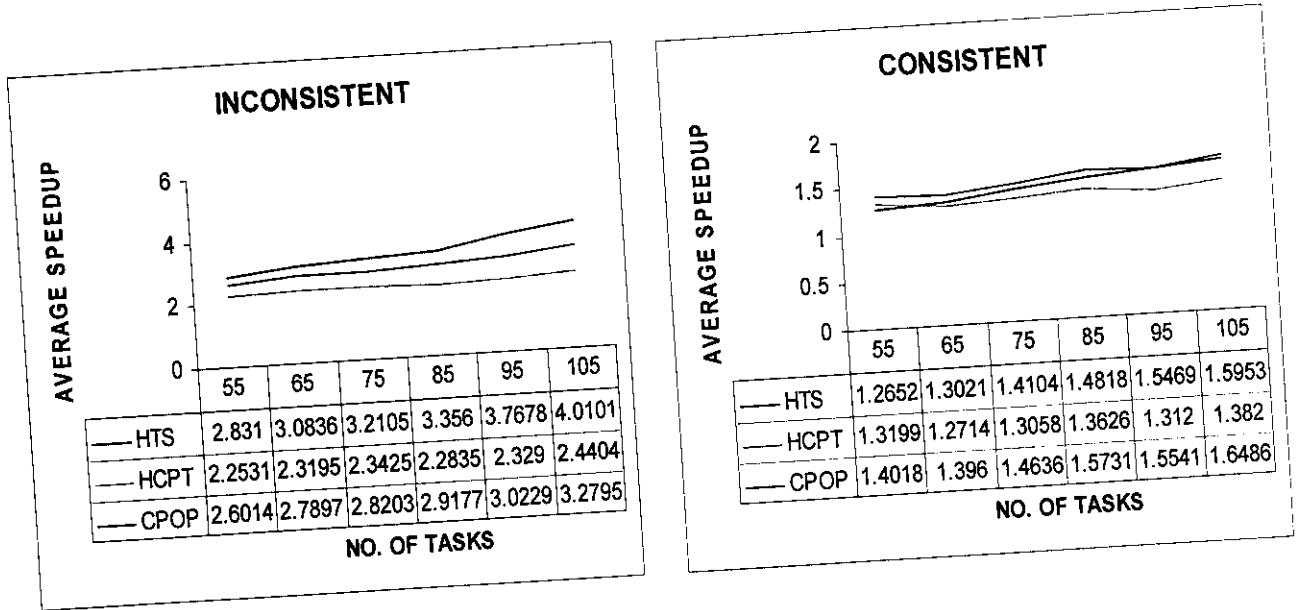


Figure 4.13

Figure 4.14

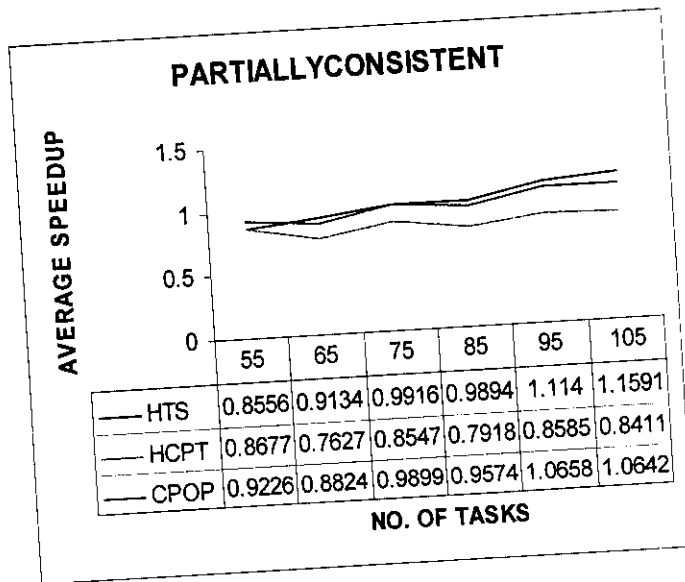


Figure 4.15

Low Task Heterogeneity High Machine Heterogeneity

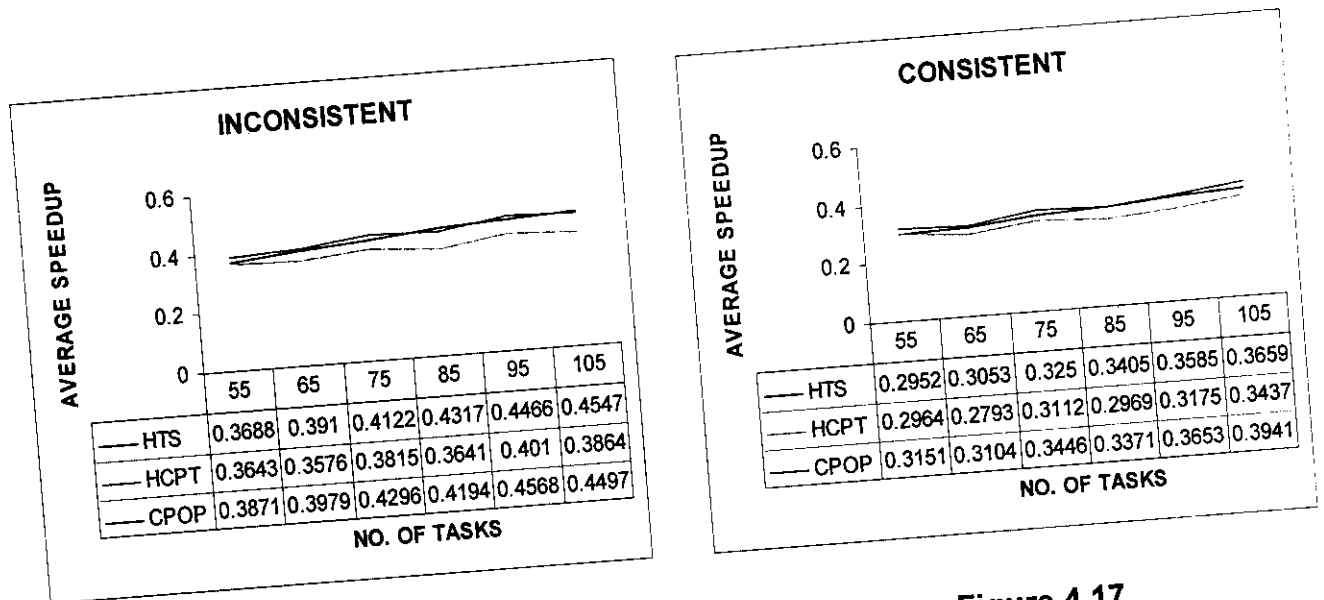


Figure 4.16

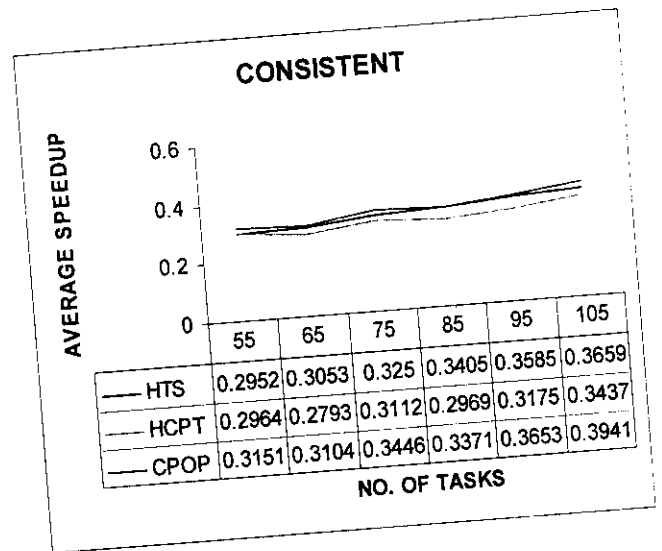


Figure 4.17

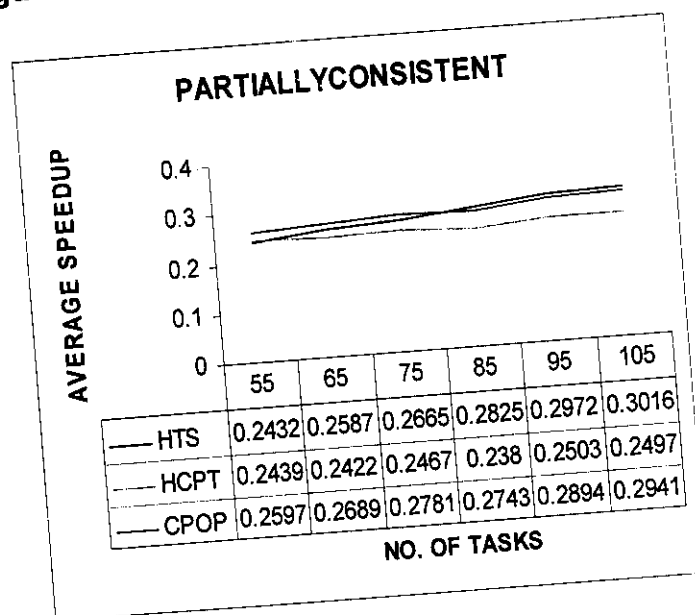


Figure 4.18

High Task Heterogeneity Low Machine Heterogeneity

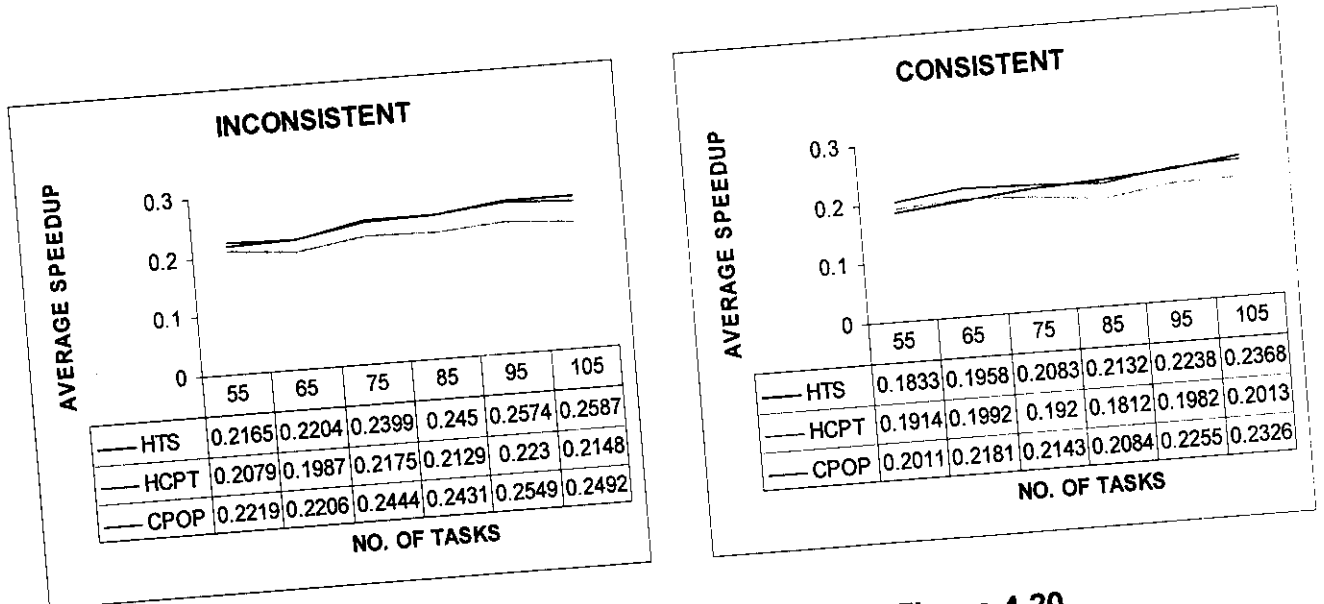


Figure 4.19

Figure 4.20

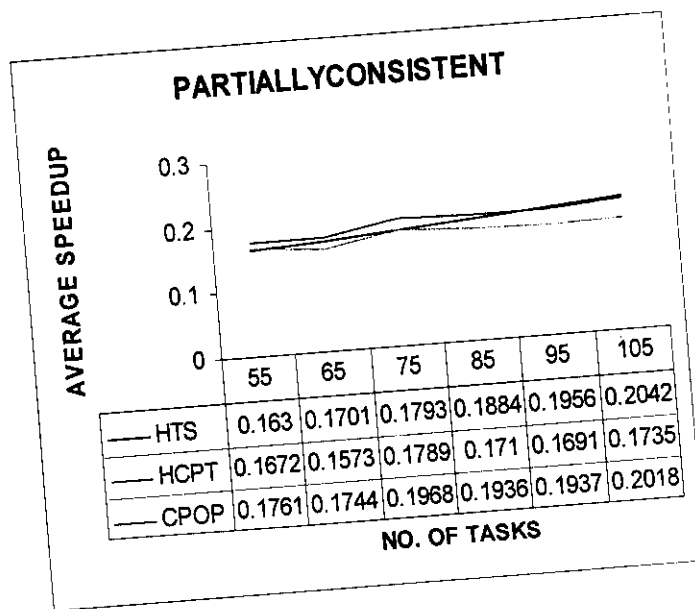


Figure 4.21

High Task Heterogeneity High Machine Heterogeneity

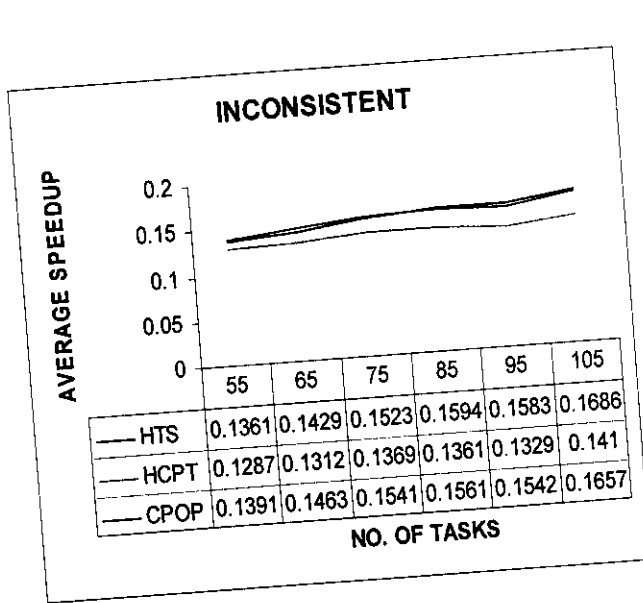


Figure 4.22

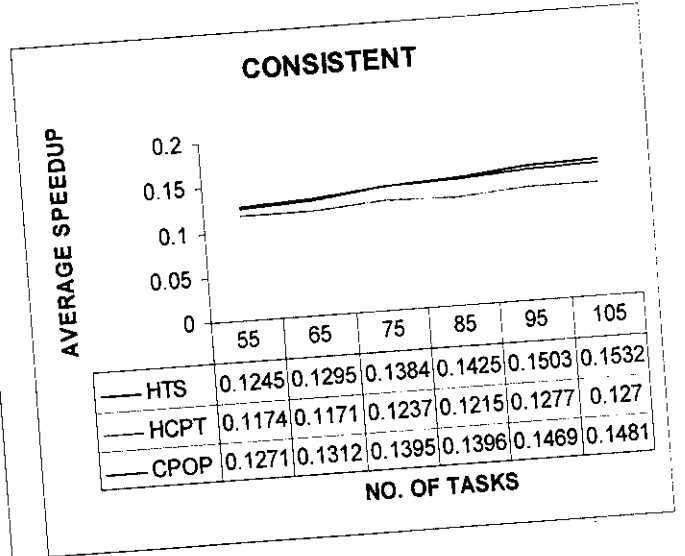


Figure 4.23

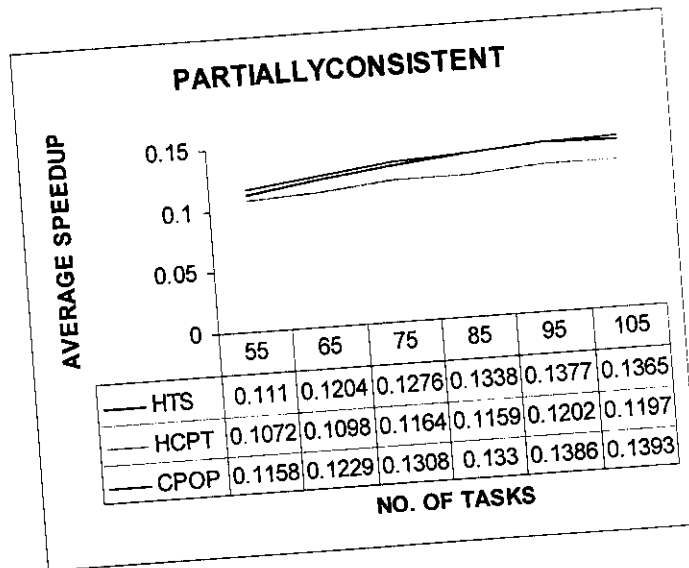


Figure 4.24

4.1.3 Number of Favorable Cases for 1000 Trials

4.1.3.1 Low Task Heterogeneity Low Machine Heterogeneity

Inconsistent

| NO. OF NODES | METHODS | | |
|--------------|---------|------|------|
| | HTS | HCPT | CPOP |
| 55 | 490 | 46 | 294 |
| 65 | 573 | 20 | 256 |
| 75 | 646 | 15 | 223 |
| 85 | 742 | 7 | 151 |
| 95 | 786 | 6 | 104 |
| 105 | 844 | 2 | 83 |

Table 4.1

Consistent

| NO. OF NODES | METHODS | | |
|--------------|---------|------|------|
| | HTS | HCPT | CPOP |
| 55 | 376 | 66 | 354 |
| 65 | 468 | 40 | 308 |
| 75 | 524 | 30 | 274 |
| 85 | 661 | 15 | 166 |
| 95 | 739 | 10 | 149 |
| 105 | 806 | 8 | 89 |

Table 4.2

Partially Consistent

| NO. OF NODES | METHODS | | |
|--------------|---------|------|------|
| | HTS | HCPT | CPOP |
| 55 | 410 | 78 | 321 |
| 65 | 571 | 26 | 259 |
| 75 | 552 | 25 | 271 |
| 85 | 722 | 11 | 164 |
| 95 | 756 | 13 | 130 |
| 105 | 833 | 10 | 79 |

Table 4.3

4.1.3.2 Low Task Heterogeneity High Machine Heterogeneity

Inconsistent

| NO. OF NODES | METHODS | | |
|-----------------|---------|------|------|
| | HTS | HCPT | CPOP |
| 55 | 363 | 106 | 485 |
| 65 | 496 | 36 | 436 |
| 75 | 453 | 29 | 482 |
| 85 | 672 | 15 | 282 |
| 95 | 667 | 23 | 278 |
| 105 | 735 | 13 | 232 |

Table 4.4

Consistent

| NO. OF NODES | METHODS | | |
|-----------------|---------|------|------|
| | HTS | HCPT | CPOP |
| 55 | 389 | 117 | 457 |
| 65 | 477 | 48 | 444 |
| 75 | 504 | 50 | 415 |
| 85 | 663 | 24 | 284 |
| 95 | 682 | 20 | 272 |
| 105 | 750 | 14 | 213 |

Table 4.5

Partially Consistent

| NO. OF NODES | METHODS | | |
|-----------------|---------|------|------|
| | HTS | HCPT | CPOP |
| 55 | 379 | 95 | 472 |
| 65 | 479 | 47 | 433 |
| 75 | 530 | 34 | 400 |
| 85 | 661 | 27 | 290 |
| 95 | 675 | 19 | 286 |
| 105 | 727 | 9 | 245 |

Table 4.6

4.1.3.3 High Task Heterogeneity Low Machine Heterogeneity

Inconsistent

| NO. OF NODES | METHODS | | |
|--------------|---------|------|------|
| | HTS | HCPT | CPOP |
| 55 | 406 | 86 | 473 |
| 65 | 448 | 40 | 480 |
| 75 | 464 | 31 | 465 |
| 85 | 646 | 20 | 312 |
| 95 | 645 | 22 | 295 |
| 105 | 711 | 24 | 248 |

Table 4.7

Consistent

| NO. OF NODES | METHODS | | |
|--------------|---------|------|------|
| | HTS | HCPT | CPOP |
| 55 | 396 | 112 | 448 |
| 65 | 458 | 49 | 451 |
| 75 | 525 | 40 | 405 |
| 85 | 676 | 16 | 288 |
| 95 | 690 | 28 | 251 |
| 105 | 758 | 18 | 198 |

Table 4.8

Partially Consistent

| NO. OF NODES | METHODS | | |
|--------------|---------|------|------|
| | HTS | HCPT | CPOP |
| 55 | 393 | 137 | 418 |
| 65 | 457 | 49 | 457 |
| 75 | 496 | 56 | 412 |
| 85 | 659 | 22 | 287 |
| 95 | 686 | 24 | 270 |
| 105 | 764 | 18 | 194 |

Table 4.9

4.1.3.4 High Task Heterogeneity High Machine Heterogeneity

Inconsistent

| NO. OF NODES | METHODS | | |
|--------------|---------|------|------|
| | HTS | HCPT | CPOP |
| 55 | 360 | 108 | 502 |
| 65 | 451 | 37 | 466 |
| 75 | 470 | 42 | 465 |
| 85 | 622 | 29 | 330 |
| 95 | 659 | 29 | 302 |
| 105 | 659 | 29 | 302 |

Table 4.10

Consistent

| NO. OF NODES | METHODS | | |
|--------------|---------|------|------|
| | HTS | HCPT | CPOP |
| 55 | 421 | 104 | 452 |
| 65 | 490 | 47 | 438 |
| 75 | 530 | 44 | 411 |
| 85 | 672 | 29 | 276 |
| 95 | 691 | 32 | 267 |
| 105 | 749 | 19 | 215 |

Table 4.11

Partially Consistent

| NO. OF NODES | METHODS | | |
|--------------|---------|------|------|
| | HTS | HCPT | CPOP |
| 55 | 414 | 112 | 448 |
| 65 | 454 | 43 | 471 |
| 75 | 490 | 50 | 439 |
| 85 | 656 | 18 | 306 |
| 95 | 615 | 33 | 325 |
| 105 | 711 | 16 | 249 |

Table 4.12

4.2 EXPERIMENTAL RESULTS AND DISCUSSIONS FOR INDEPENDENT JOBS

This section presents performances comparison of the proposed algorithm with the existing CPOP and HCPT algorithms.

4.2.1. Makespan Comparison – Independent Jobs

Low Task Heterogeneity Low Machine Heterogeneity

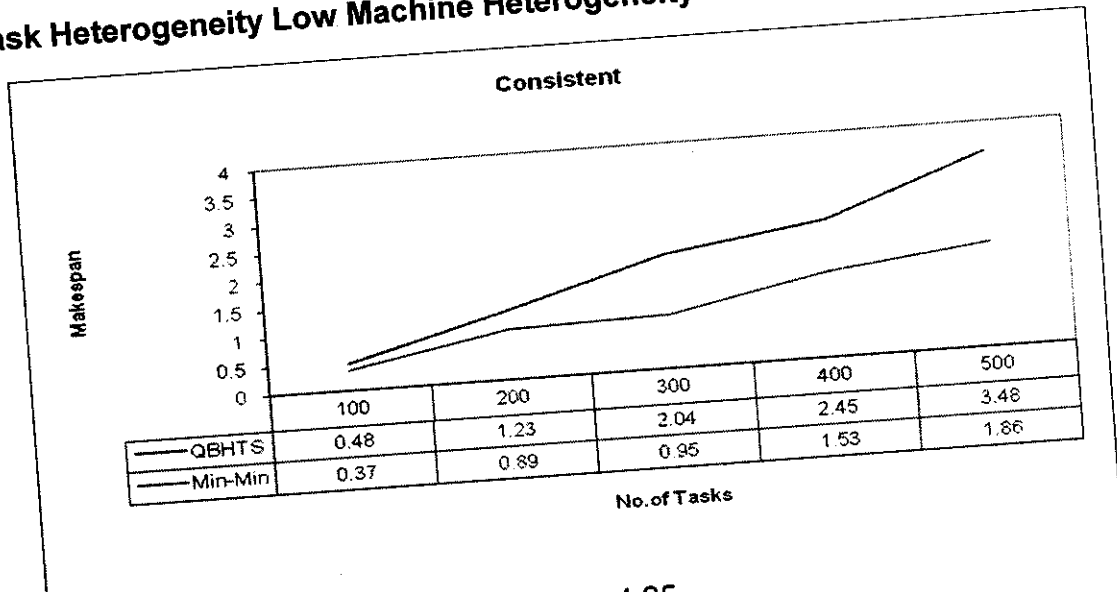


Figure 4.25

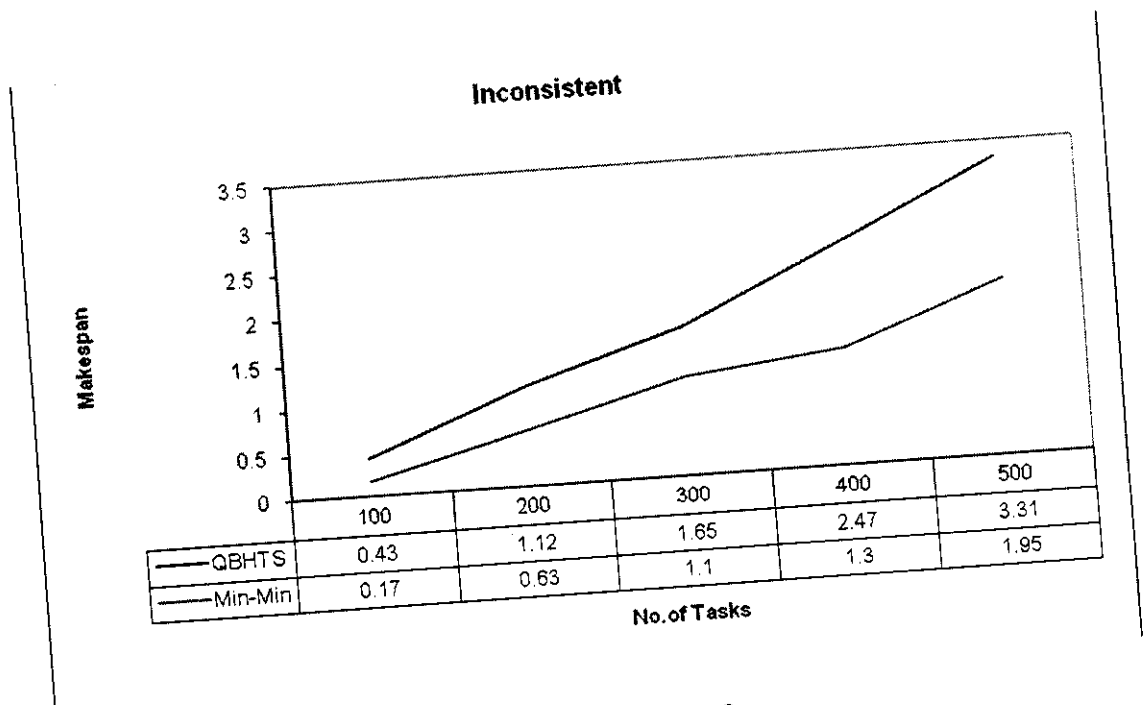


Figure 4.26

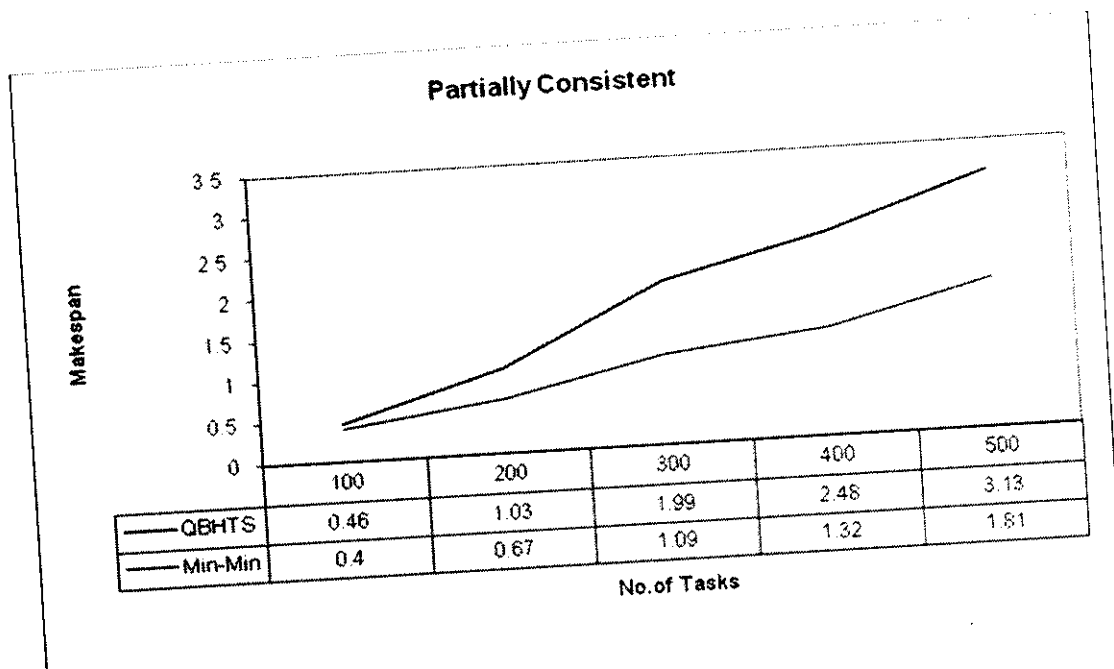


Figure 4.27

Low Task Heterogeneity High Machine Heterogeneity

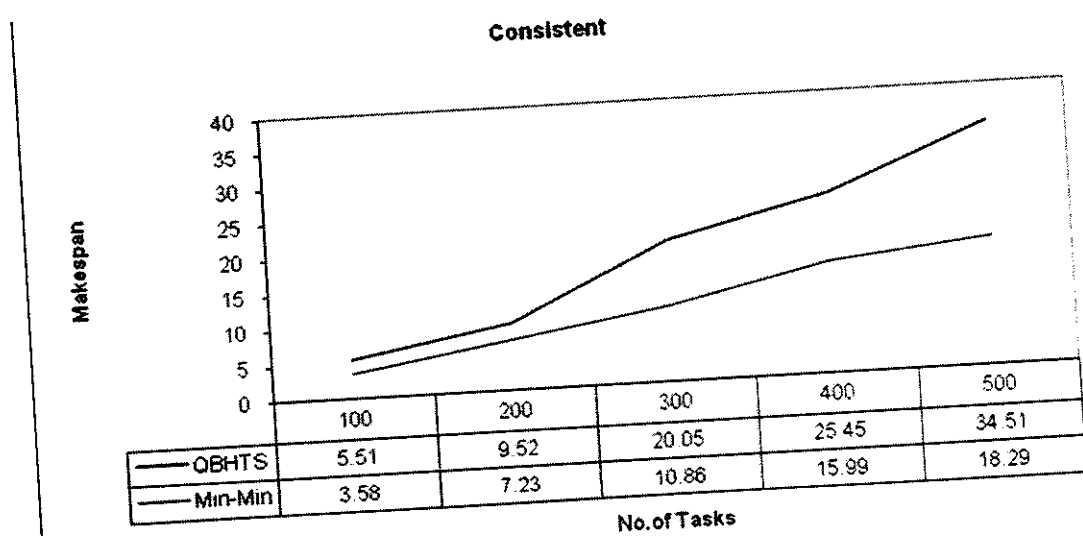


Figure 4.28

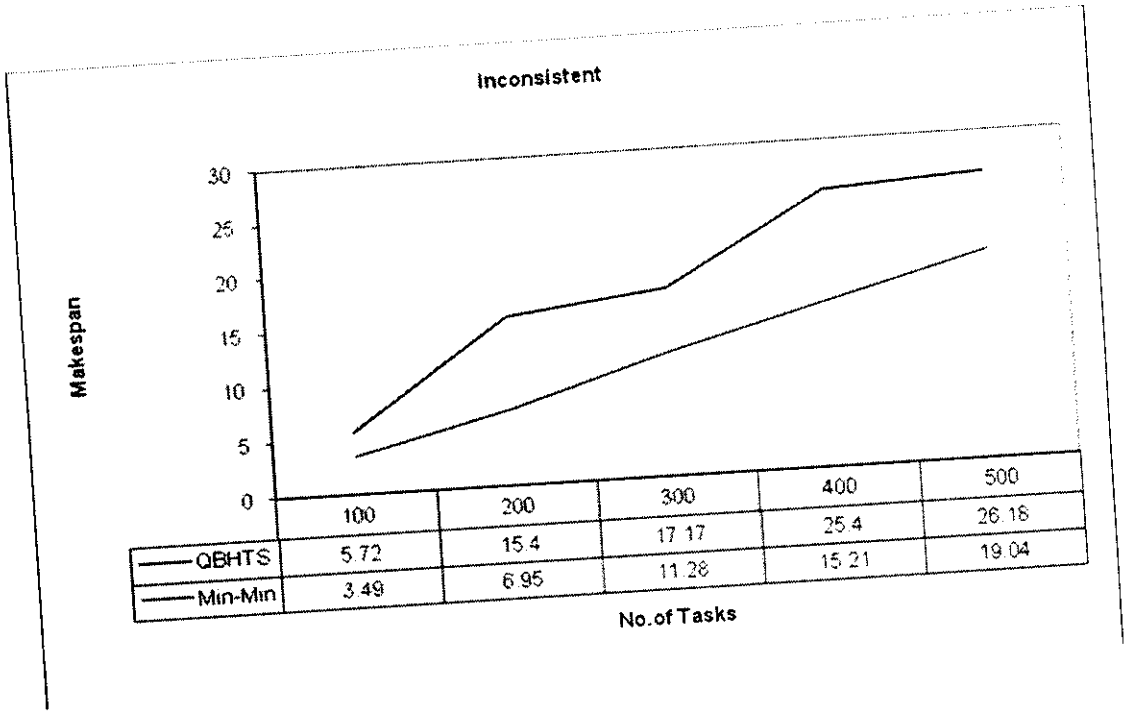


Figure 4.29

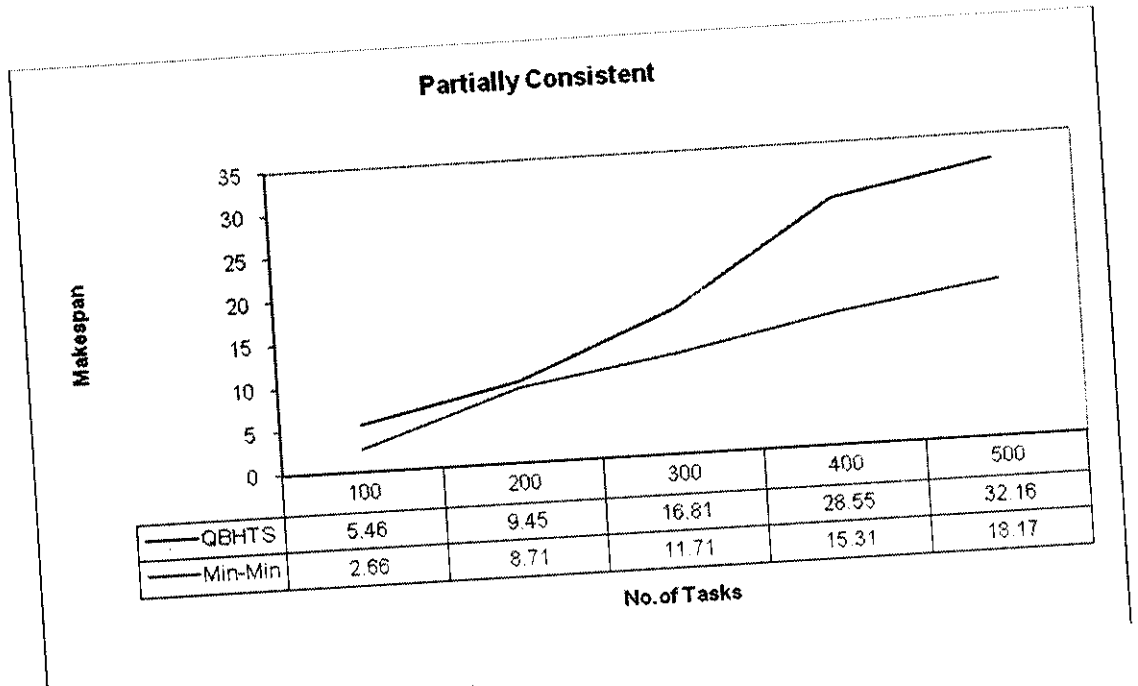


Figure 4.30

High Task Heterogeneity Low Machine Heterogeneity

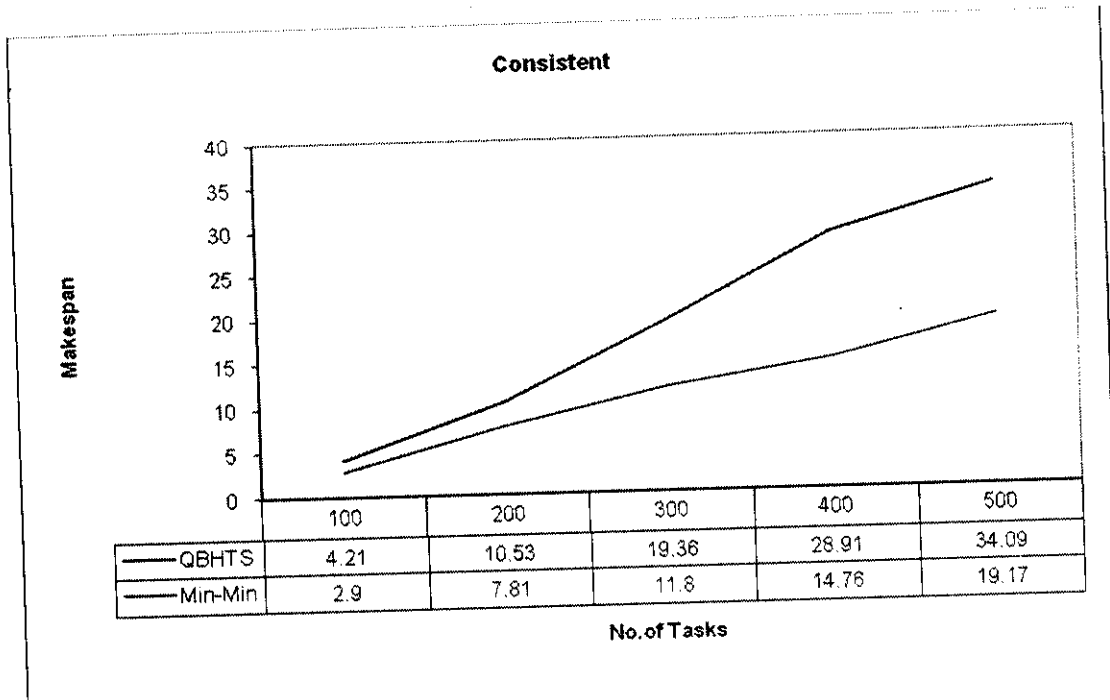


Figure 4.31

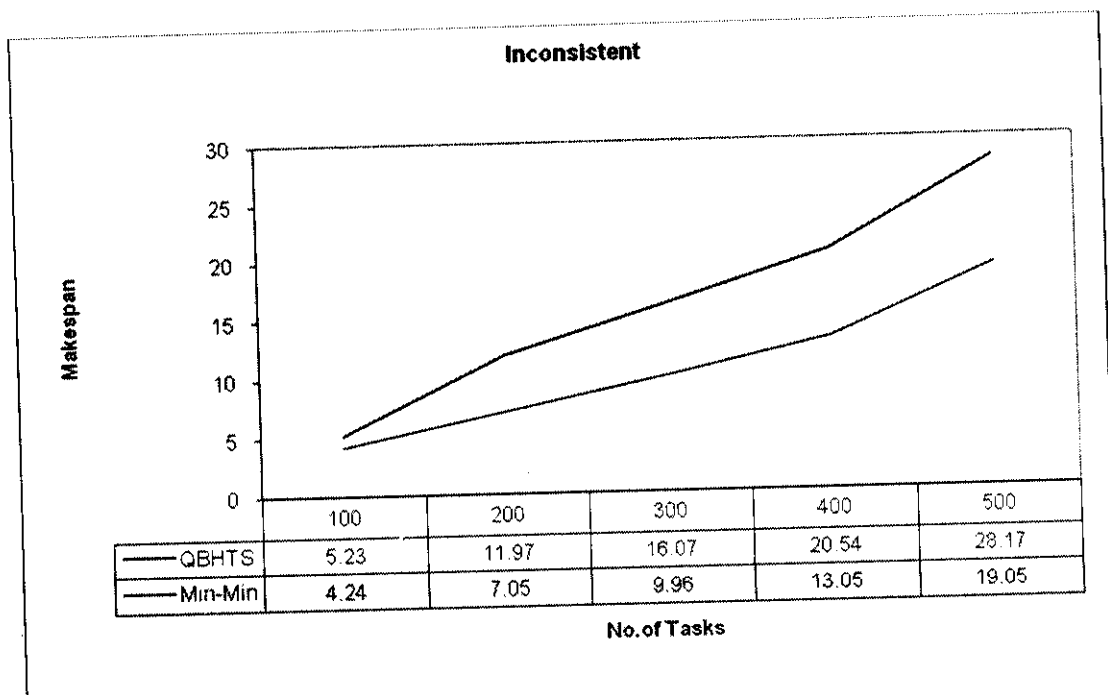


Figure 4.32

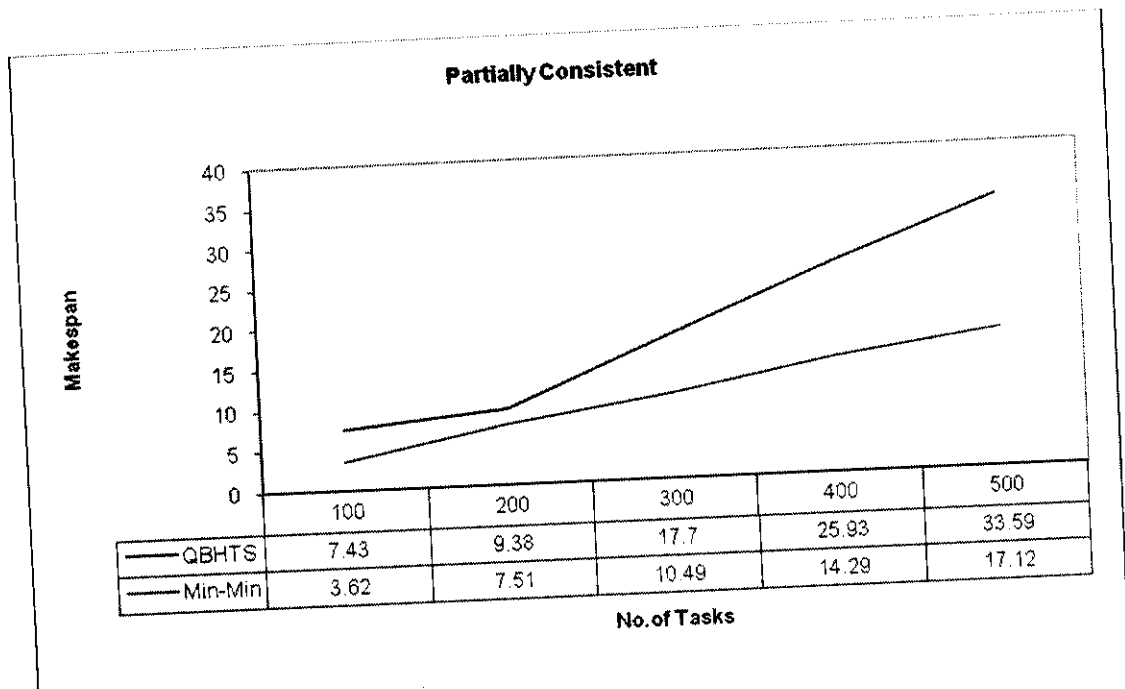


Figure 4.33

High Task Heterogeneity High Machine Heterogeneity

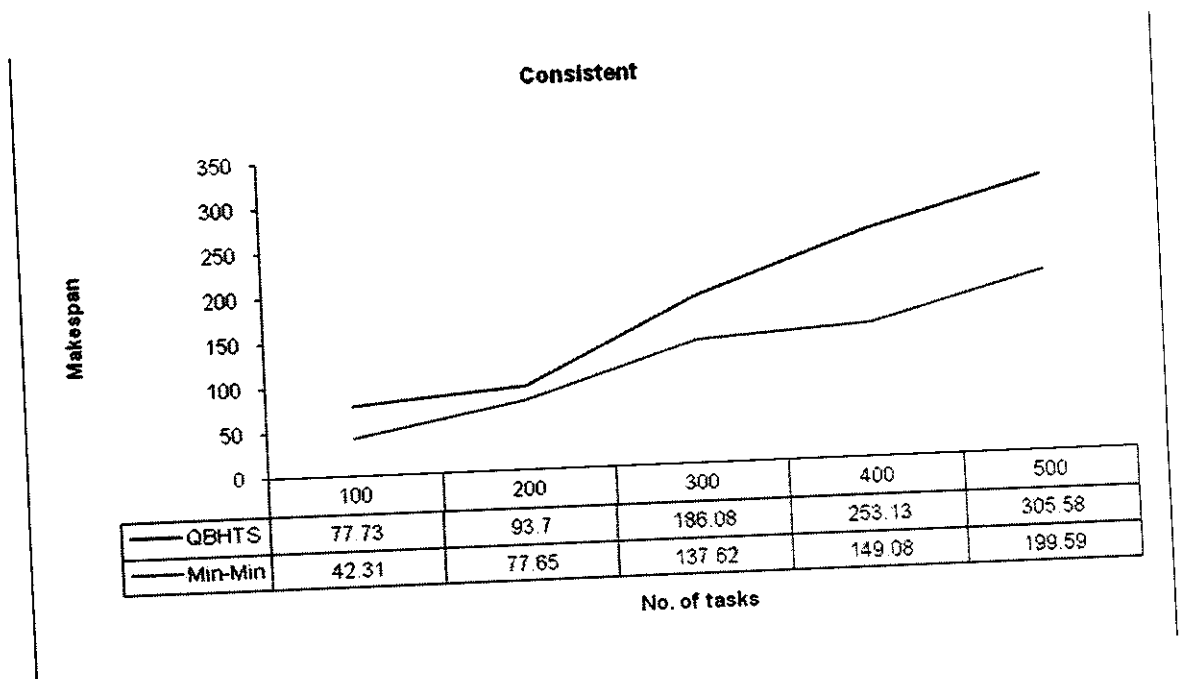


Figure 4.34

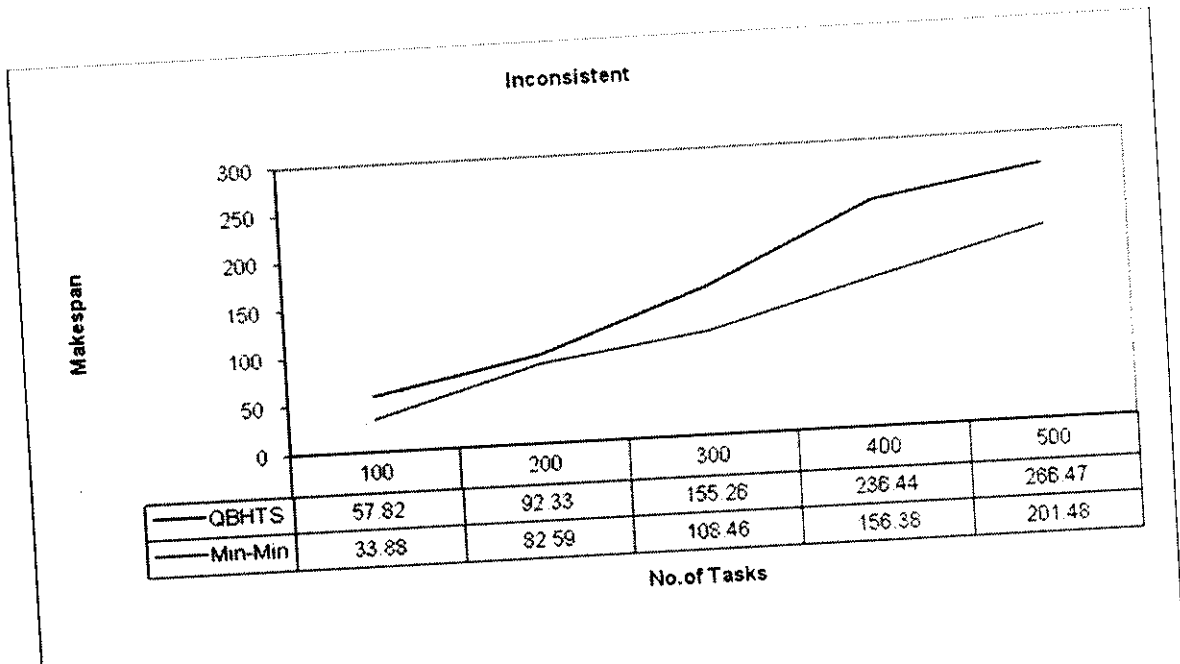


Figure 4.35

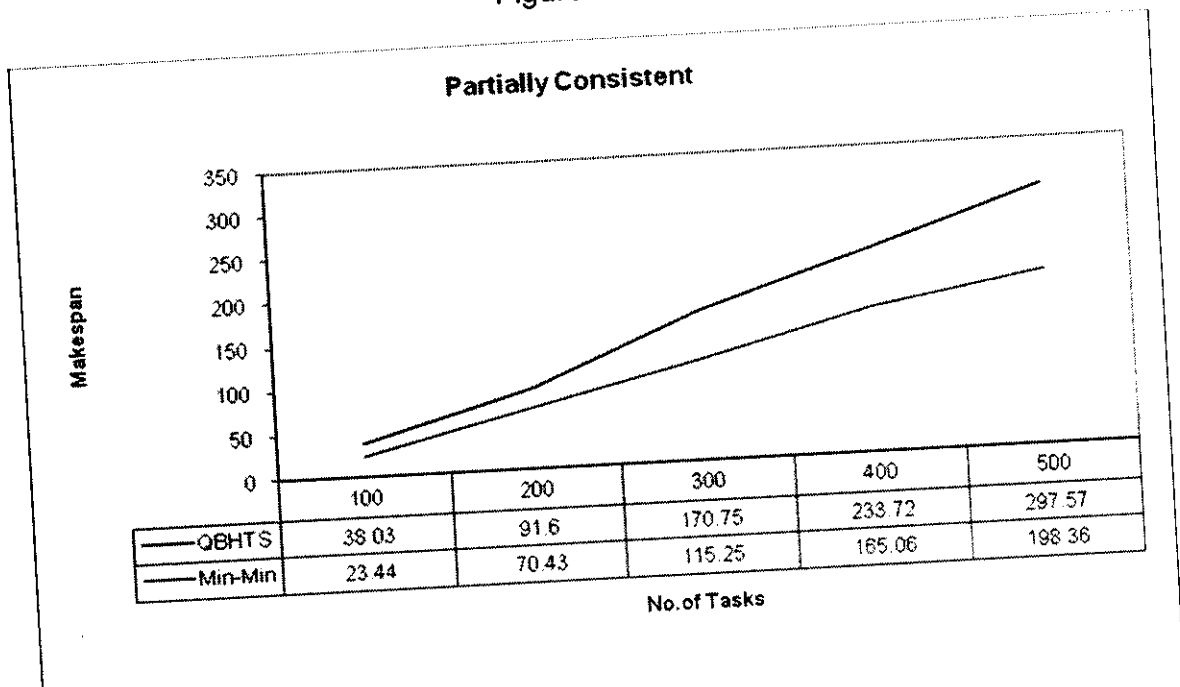


Figure 4.36

CHAPTER 5

CONCLUSION AND FUTURE ENHANCEMENTS

In this project, the **Heterogeneous Task Scheduling (HTS)** algorithm for scheduling tasks onto any number of heterogeneous machines is presented. Based on the experimental study using a large set (60K) of randomly generated application graphs with various characteristics, the HTS outperformed the other algorithms in terms of performance, complexity, running time and cost metrics including speedup, frequency of best results and average Makespan. Because of its robust performance, low running time, and the ability to give stable performance over a wide range of graph structures, the HTS algorithm is a viable solution for the DAG scheduling problem with **higher number of nodes**, on heterogeneous systems.

This work also studied of the **QoS Based Heterogeneous Task Scheduling (QBHTS)** algorithm which shows better resource matching and requirement satisfaction. Even though the Makespan is higher in our work, the graph shows that the increase in Makespan does not increase much for all the combinations of Task and Machine heterogeneity. For further work, this work can be improved to decrease the Makespan by considering the suffrage values [12].

APPENDIX

Code for HTS

```

//AVERAGE WEIGHT CALCULATION
for(i=0;i<nodes_count;i++)
{
    for(j=0;j<machine_count;j++)
        tot[i]+=weight[i][j];
    avg[i]=tot[i]/machine_count;
}
for(i=0;i<nodes_count;i++)
{
    for(j=0;j<parent_length[i];j++)
        comm_cost[i][j]=(avg[i]+avg[parent[i][j]])/2;
}
for(i=0;i<nodes_count;i++)
{
    for(j=0;j<children_length[i];j++)
        comm_cost_child[i][j]=(avg[i]+avg[children[i][j]])/2;
}
min=weight[0][0];
for(i=0;i<2;i++)
{
    if(weight[0][i+1]<min)
    {
        min=weight[0][i+1];
        current_machine=i+1;
    }
}
available_time[current_machine]=(int)min;
parent_finishtime[0]=(int)min;
max_parent[0]=0;          machine[0]=2;
for(i=0;i<children_length[0];i++)
    list.add(children[0][i]);
while(!list.isEmpty())
{
    list1.addAll(list);
    while(!list1.isEmpty())
    {

```

```

in1=(Integer)list1.get(0);
n=in1.intValue();
current_machine=(int)machine[max_parent[n]];
for(j=0;j<machine_count;j++)
{
    x=(int)available_time[j];
    flag=1;
    if(parent_length[n]==1)
    {
        k=0;
        if(machine[parent[n][0]==j)
            flag=0;
        else
            flag=1;
    }

temp_var1[k]=parent_finishtime[parent[n][k]]+(flag*(int)comm_cost[n][k]);

temp_var[k]=Math.max(x,temp_var1[k])+(int)weight[n][j];
    }
    if(parent_length[n]>1)
    {
        for(k=0;k<parent_length[n];k++)
        {
            if(machine[parent[n][k]==j)
                flag=0;
            else
                flag=1;
        }

temp_var1[k]=parent_finishtime[parent[n][k]]+(flag*(int)comm_cost[n][k]);

temp_var[k]=Math.max(x,temp_var1[k])+(int)weight[n][j];
    }
    }
    flag=0;
    max=temp_var[0];
    index1=0;
    for(k=0;k<parent_length[n]-1;k++)
    {
        if(temp_var[k+1]>max)
        {
            max=temp_var[k+1];
            index1=k+1;
        }
    }
    temp_eeft[j]=(int)max;
    max_parent[n]=index1;
}

```

```

index=0;
m=0;
min=temp_eeft[0];
for(l=0;l<machine_count;l++)
{
    if(temp_eeft[l]<min)
    {
        min=temp_eeft[l];
        index=l;
        m=index;
    }
}
list_temp1.add(index);
list_temp.add((int)min);
list1.remove(0);
}
index=0;    m=0;
in1=(Integer)list_temp.get(0);
min=in1.intValue();
for(i=0;i<list_temp.size();i++)
{
    in1=(Integer)list_temp.get(i);
    tmp=in1.intValue();
    if(tmp<min)
    {
        min=tmp;
        index=i;
        m=index;
    }
}
in=(Integer)list_temp1.get(index);
mac=in.intValue();
current_machine=mac;
in=(Integer)list.get(index);
task=in.intValue();
machine[task]=current_machine;
in=(Integer)list_temp.get(index);
time=in.intValue();
EEFT[task]=time;
available_time[current_machine]=time;
parent_finishtime[task]=time;
l=list.indexOf(task);
list.remove(l);
ready_list.add(task);
flag=0;

for(j=0;j<children_length[task];j++)
{

```

```

    for(k=0;k<parent_length[children[task][j]];k++)
    {
        if(ready_list.contains(parent[children[task][j]][k]))
        {
            count++;
        }
    }
    if(n!=nodes_count-1)
    {
        if(count==parent_length[children[task][j]])
            list.add(children[task][j]);
    }
    count=0;
}
list_temp.clear();
list_temp1.clear();
}

```

Code for HCPT

//AVERAGE WEIGHT CALCULATION

```

for(i=0;i<nodes_count;i++)
{
    for(j=0;j<machine_count;j++)
        tot[i]+=weight[i][j];
    avg[i]=tot[i]/machine_count;
}
for(i=0;i<nodes_count;i++)
{
    for(j=0;j<parent_length[i];j++)
        comm_cost[i][j]=(avg[i]+avg[parent[i][j]])/2;
}
for(i=0;i<nodes_count;i++)
{
    for(j=0;j<children_length[i];j++)
        comm_cost_child[i][j]=(avg[i]+avg[children[i][j]])/2;
}
//STEP 1: CALCULATE AEST

```

```

for(i=1;i<nodes_count;i++)
{
    for(j=0;j<parent_length[i];j++)
    {
        x=parent[i][j];
        temp[j]=AEST[x]+avg[x]+comm_cost[i][j];
    }
}

```

```

    }
    max=temp[0];
    for(l=0;l<parent_length[i];l++)
        if(temp[l]>max)
            max=temp[l];
    AEST[++k]=max;
    max=0.0f;
}

//STEP 2:CALCULATE ALST

ALST[nodes_count-1]=AEST[nodes_count-1];
k=nodes_count-1;
for(i=nodes_count-2;i>=0;i--)
{
    for(j=0;j<children_length[i];j++)
    {
        x=children[i][j];
        minimum[j]=ALST[x]-comm_cost_child[i][j];
    }

    min=minimum[0];
    for(l=0;l<children_length[i];l++)
        if(minimum[l]<min)
            min=minimum[l];
    ALST[--k]=(min-avg[i]);
}
ALST[0]=0;
for(i=0;i<nodes_count;i++);
////System.out.println(ALST[i]);
//STEP 3:LISTING PHASE

k=0;
/*****PUSHING THE CRITICAL NODES ON THE STACK*****/
for(i=nodes_count-1;i>=0;i--)
{
    if((double)ALST[i]==(double)AEST[i])
    {
        //System.out.println(i);
        critical_nodes.push(i);
    }
}
list.add(0,critical_nodes.peek());

/*****WHILE STACK IS NOT EMPTY*****/

while(!critical_nodes.empty())
{
    count=0;
    i=critical_nodes.peek().intValue();

```

```

for(j=0;j<(parent_length[i]);j++)
{
    /*****IF THERE IS AN UNLISTED PARENT OF TOS*****/
    if(!list.contains(parent[i][j]))
        critical_nodes.push(parent[i][j]);
}
count=0;
i=critical_nodes.peek().intValue();
for(k=0;k<parent_length[i];k++)
{
    if(list.contains(parent[i][k]))
        count++;
}
if(count==parent_length[i])
{
    list.add(critical_nodes.pop());
    count=0;
}

    /*****POP THE TOS AND ENQUEUE IT IN THE LIST*****/
}
list.remove(0);

    /*****OUTPUT OF LISTING PHASE*****/

for(i=0;i<list.size();i++);
///System.out.println(list.get(i));

    /*ASSIGNMENT PHASE*/
m=0;
min=weight[0][0];
for(i=0;i<machine_count-1;i++)
{
    if(weight[0][i+1]<min)
    {
        min=weight[0][i+1];
        m=i+1;
    }
}
available_time[m]=(int)min;
parent_finishtime[0]=(int)min;
for(i=1;i<list.size();i++)
{
    in=(Integer)list.get(i);
    n=in.intValue();
    for(j=0;j<machine_count;j++)
    {
        flag=0;
    }
}

```

```

if(m==j)
    flag=0;
else
    flag=1;
x=(int)available_time[j];
for(k=0;k<parent_length[n];k++)
{
    if(parent_length[n]>1)
    {
        if(machine[parent[n][k]]==j)
            flag=0;
        else
            flag=1;
    }
temp_var1[k]=parent_finishtime[parent[n][k]]+(flag*(int)comm_cost[n][k]);
temp_var[k]=Math.max(x,temp_var1[k])+(int)weight[n][j];
}
flag=0;
max=temp_var[0];
index1=0;
for(k=0;k<parent_length[n]-1;k++)
{
    if(temp_var[k+1]>max)
    {
        max=temp_var[k+1];
        index1=k+1;
    }
}
temp_eeft[j]=(double)max;
a[j]=index1;
}
index=0;    m=0;
min=temp_eeft[0];
for(l=0;l<machine_count-1;l++)
{
    if(temp_eeft[l+1]<min)
    {
        min=temp_eeft[l+1];
        index=l+1;
        m=index;
    }
}
machine[n]=m;
available_time[index]=(int)min;
EEFT[n]=(double)min;
parent_finishtime[n]=(int)min;
}

```


Code for CPOP

//AVERAGE WEIGHT CALCULATION

```

for(i=0;i<nodes_count;i++)
{
    for(j=0;j<machine_count;j++)
        tot[i]+=weight[i][j];
    avg[i]=tot[i]/machine_count;
}

for(i=0;i<nodes_count;i++)
{
    for(j=0;j<parent_length[i];j++)
        comm_cost[i][j]=(avg[i]+avg[parent[i][j]])/2;
}

for(i=0;i<nodes_count;i++)
{
    for(j=0;j<children_length[i];j++)
        comm_cost_child[i][j]=(avg[i]+avg[children[i][j]])/2;
}

for(i=1;i<nodes_count;i++)
{
    for(j=0;j<parent_length[i];j++)
    {
        x=parent[i][j];
        temp[j]=down_rank[x]+avg[x]+comm_cost[i][j];
    }
    max=temp[0];
    for(l=0;l<parent_length[i];l++)
        if(temp[l]>max)
            max=temp[l];
    down_rank[++k]=max;
    max=0.0f;
}
up_rank[nodes_count-1]=avg[nodes_count-1];
k=nodes_count-1;
for(i=nodes_count-2;i>=0;i--)
{
    for(j=0;j<children_length[i];j++)
    {

```

```

        x=children[i][j];
        maximum[j]=up_rank[x]+comm_cost_child[i][j];
    }
    max=maximum[0];
    for(l=0;l<children_length[i];l++)
        if(maximum[l]>max)
            max=maximum[l];
    up_rank[-k]=(max+avg[i]);
}
for(i=0;i<nodes_count;i++)
{
    priority[i]=up_rank[i]+down_rank[i];
}
i=0;
critical_path[0]=0;
while(critical_path[i]!=nodes_count-1)
{
    max=priority[critical_path[i]];
    index=children[critical_path[i]][0];
    for(j=0;j<children_length[critical_path[i]];j++)
    {
        if(priority[children[critical_path[i]][j]]>=max)
        {
            max=priority[children[critical_path[i]][j]];
            index=children[critical_path[i]][j];
        }
    }
    critical_path[++i]=index;
}
//selecting cpp
for(i=0;i<machine_count;i++)
{
    tmp=0;
    for(j=0;j<critical_path.length;j++)
        tmp=tmp+(int)weight[critical_path[j]][i];
    temp[i]=tmp;
}
min=temp[0];
for(i=0;i<machine_count;i++)
{
    if(temp[i]<min)
    {
        min=temp[i];
        cpp=i;
    }
}
list.add(0,0);

```

```

m=cpp;
available_time[m]=(int)weight[0][cpp];
parent_finishtime[0]=(int)weight[0][cpp];
EEFT[0]=(int)weight[0][cpp];
while(!list.isEmpty())
{
    in=(Integer)list.get(0);
    tmp=in.intValue();
    max=priority[tmp];
    index=tmp;
    for(i=0;i<list.size();i++)
    {
        in=(Integer)list.get(i);
        tmp=in.intValue();
        if(priority[tmp]>=max)
        {
            max=priority[tmp];
            index=i;
        }
    }
    in=(Integer)list.get(index);
    n=in.intValue();
    if(n!=0)
    {
        for(j=0;j<machine_count;j++)
        {
            flag=0;
            if(m==j)
                flag=0;
            else
                flag=1;

            x=available_time[j];
            for(k=0;k<parent_length[n];k++)
            {
                if(parent_length[n]>1)
                {
                    if(machine[parent[n]][k]==j)
                        flag=0;
                    else
                        flag=1;
                }
            }
            temp_var1[k]=parent_finishtime[parent[n]][k]+(flag*(int)comm_cost[n][k]);
            temp_var[k]=Math.max(x,temp_var1[k])+(int)weight[n][j];
        }
        flag=0;
        max=temp_var[0];
    }
}

```

```

index1=0;

for(k=0;k<parent_length[n]-1;k++)
{
    if(temp_var[k+1]>max)
    {
        max=temp_var[k+1];
        index1=k+1;
    }
}
temp_eeft[j]=(int)max;
a[j]=index1;
}
flag=0;
for(j=0;j<critical_path.length;j++)
{
    if(n==critical_path[j])
        flag=1;
}
if(flag==1)
{
    m=cpp;
    min=temp_eeft[cpp];
    machine[n]=m;
    available_time[m]=(int)min;
    EEFT[n]=(int)min;
    parent_finishtime[n]=(int)min;
}
else
{
    index=0;
    m=0;
    min=temp_eeft[0];
    for(l=0;l<machine_count;l++)
    {
        if(temp_eeft[l]<min)
        {
            min=temp_eeft[l];
            index=l;
            m=index;
        }
    }
    machine[n]=m;
    available_time[index]=(int)min;
    EEFT[n]=(int)min;
    parent_finishtime[n]=(int)min;
}
}
}

```

```

l=list.indexOf(n);
list.remove(l);
ready_list.add(n);
flag=0;
for(j=0;j<children_length[n];j++)
{
    for(k=0;k<parent_length[children[n][j]];k++)
    {
        if(ready_list.contains(parent[children[n][j]][k]))
            count++;
    }
    if(n!=nodes_count-1)
    {
        if(count==parent_length[children[n][j]])
            list.add(children[n][j]);
    }
    count=0;
}
}

```

Code for QoS based scheduling

```

/*.....Makespan calculation for QBHTS Algorithm.....*/
for(int i=0;i<task;i++)
{
    op=(int)mmat[i][1];
    switch(op)
    {
        case 0:
            m[0]=ob.makespan(m[0],i,0,cstmat);
            break;
        case 1:
            m[1]=ob.makespan(m[1],i,1,cstmat);
            break;
        case 2:
            m[2]=ob.makespan(m[2],i,2,cstmat);
            break;
        case 3:
            m[3]=ob.makespan(m[3],i,3,cstmat);
            break;
        case 4:
            m[4]=ob.makespan(m[4],i,4,cstmat);
            break;
        case 5:
            m[5]=ob.makespan(m[5],i,5,cstmat);
            break;
        case 6:
            m[6]=ob.makespan(m[6],i,6,cstmat);

```

```

        break;
    case 7:
        m[7]=ob.makespan(m[7],i,7,cstmat);
        break;
    case 8:
        m[8]=ob.makespan(m[8],i,8,cstmat);
        break;
    case 9:
        m[9]=ob.makespan(m[9],i,9,cstmat);
        break;
    case 10:
        m[10]=ob.makespan(m[10],i,10,cstmat);
        break;
    case 11:
        m[11]=ob.makespan(m[11],i,11,cstmat);
        break;
    case 12:
        m[12]=ob.makespan(m[12],i,12,cstmat);
        break;
    case 13:
        m[13]=ob.makespan(m[13],i,13,cstmat);
        break;
    case 14:
        m[14]=ob.makespan(m[14],i,14,cstmat);
        break;
    case 15:
        m[15]=ob.makespan(m[15],i,15,cstmat);
        break;
    }
}
for(int i=0;i<16;i++)
    System.out.println("Machine "+i+" : " +m[i]);
/*..... Finding Maximum Makespan.....*/
max=m[0];
for(int i=1;i<16;i++)
    if(m[i]>max)
        max=m[i];

System.out.println("Make span for QBHTS :"+max);
for(int i=0;i<16;i++)
    m[i]=0;
/*.....Makespan calculation for Min-Min Algorithm.....*/
for(int i=0;i<task;i++)
{
    op=(int)mimat[i][1];
    switch(op)
    {
        case 0:

```

```
        m[0]=ob.makespan(m[0],i,0,cstmat);
        break;
    case 1:
        m[1]=ob.makespan(m[1],i,1,cstmat);
        break;
    case 2:
        m[2]=ob.makespan(m[2],i,2,cstmat);
        break;
    case 3:
        m[3]=ob.makespan(m[3],i,3,cstmat);
        break;
    case 4:
        m[4]=ob.makespan(m[4],i,4,cstmat);
        break;
    case 5:
        m[5]=ob.makespan(m[5],i,5,cstmat);
        break;
    case 6:
        m[6]=ob.makespan(m[6],i,6,cstmat);
        break;
    case 7:
        m[7]=ob.makespan(m[7],i,7,cstmat);
        break;
    case 8:
        m[8]=ob.makespan(m[8],i,8,cstmat);
        break;
    case 9:
        m[9]=ob.makespan(m[9],i,9,cstmat);
        break;
    case 10:
        m[10]=ob.makespan(m[10],i,10,cstmat);
        break;
    case 11:
        m[11]=ob.makespan(m[11],i,11,cstmat);
        break;
    case 12:
        m[12]=ob.makespan(m[12],i,12,cstmat);
        break;
    case 13:
        m[13]=ob.makespan(m[13],i,13,cstmat);
        break;
    case 14:
        m[14]=ob.makespan(m[14],i,14,cstmat);
        break;
    case 15:
        m[15]=ob.makespan(m[15],i,15,cstmat);
        break;
}
```

```

    }
    for(int i=0;i<16;i++)
        System.out.println("Machine "+i+" : " +m[i]);

    /*..... Finding Maximum Makespan.....*/
    max=m[0];
    for(int i=0;i<16;i++)
        if(m[i]>max)
            max=m[i];
    System.out.println("Make span for Min-Min :"+max);
}
for(int i=0;i<16;i++)
    m[i]=0;
}
}

/*..... Finding Maximum and Minimum For the Given Input Sorted Matrix .....*/
public double[][] minmax(double[][] smat,double[][] imat,int task)
{
    double tmp,ind;
    int flag=0,l=0,no=5;
    double[][] temp=new double[task][16];
    double[] arr= new double[16];
    for(int y=0;y<16;y++)
        arr[y]=25;
    for(int i=0;i<task;i++)
    {
        for(int j=0;j<16;j++)
        {
            tmp=smat[i][j]; ind=imat[i][j];
            for(int k=0;k<15;k++)
                if(ind==arr[k])
                    flag=1;
            if(flag!=1)
            {
                temp[i][0]=tmp; temp[i][1]=ind; temp[i][2]=0;
                arr[l++]=ind;
                break;
            }
            else
            {
                flag=0; continue;
            }
        }
    }
    if(i==no)
    {
        no=no+16; l=0;
        for(int y=0;y<16;y++)
    }
}

```



```

        arr[y]=25; flag=0;
    }
    else
        continue;
}
return temp;
}
/*..... FOR FINDING MAXIMUM IN A ROW.....*/
double[][] sort(double[][] mat,int task)
{
    double tmp;
    double[][] temp=new double[task][16];
    for(int i=0;i<task;i++)
        for(int j=0;j<16;j++)
            temp[i][j]=mat[i][j];
//    SORTING THE MATRIX
    for(int i=0;i<task;i++)
        for(int j=0;j<16;j++)
            for(int k=j+1;k<16;k++)
                {
                    if(temp[i][j]<temp[i][k])
                    {
                        tmp=temp[i][j];
                        temp[i][j]=temp[i][k];
                        temp[i][k]=tmp;
                    }
                }
    return temp;
}
/*..... For Finding Minimum in a row .....*/
double[][] sort1(double[][] mat,int task)
{
    double tmp;
    double[][] temp=new double[task][16];
    for(int i=0;i<task;i++)
        for(int j=0;j<16;j++)
            temp[i][j]=mat[i][j];
//    SORTING THE MATRIX
    for(int i=0;i<task;i++)
        for(int j=0;j<16;j++)
            for(int k=j+1;k<16;k++)
                {
                    if(temp[i][j]>temp[i][k])
                    {
                        tmp=temp[i][j];
                        temp[i][j]=temp[i][k];

```

```

        temp[i][k]=tmp;
    }
}
return temp;
}

```

```

/* .....Searching the index of values in one Matrix in Another Matrix .....*/

```

```

double[][] search(double[][] mat,double[][] mmat,int task)

```

```

{
    //SEARCHING THE INDEX
    double[][] temp=new double[task][16];
    double tmp;
    int tmp1=0;
    for(int i=0;i<task;i++)
    {
        for(int j=0;j<16;j++)
        {
            tmp=mmat[i][j];
            for(int k=0;k<16;k++)
            {
                if(tmp==mat[i][k])
                {
                    tmp1=k;
                    break;
                }
                else
                    continue;
            }
            temp[i][j]=tmp1;
        }
    }
    return temp;
}

```

REFERENCES

- [1] Tarek Hagra, Jan Janeček, "A Simple Scheduling Heuristic for Heterogeneous Computing Environments," Proceedings of the Second International Symposium on Parallel and Distributed Computing (ISPDC,03), 2003.
- [2] Haluk Topcuoglu, Salim Hariri, Min-You Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing", IEEE Transaction on Parallel and Distributed Systems, Vol. 13, No. 3, pp.260-274, March 2002.
- [3] T. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen, and R. Freund. "A comparison study of static mapping heuristics for a class of meta tasks on heterogeneous computing systems", 8th IEEE Heterogeneous Computing Workshop (HCW'99), pp 15-29, April 1999.
- [4] G.Sih, and E.Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures", IEEE Transaction in Parallel and Distributed Systems, Vol.4, pp.75-87, 1993.
- [5] A.Radulescu, and A.van Gemund, "Fast and Effective Task Scheduling in Heterogeneous Systems", 9th Heterogeneous Computing Workshop, pp.229-238, 2000.
- [6] I. Foster and C. Kesselman. The GRID: Blueprint for a New Computing Infrastructure, 2nd Edition, Morgan- Kaufmann, San Mateo, CA, 2004.
- [7] M. Analoui and L. Mohammad Khanli "Grid_JQA: A QoS Guided Scheduling Algorithm for Grid Computing", Proceedings of Sixth International Symposium on Parallel and distributed Systems, 2007.
- [8] Yin-Yun Shen, Xiao-Ping Li, Qian Wang, Ying-Chun Yuan., "A Hybrid QoS- Based Algorithm for Independent Tasks Scheduling in Grid" supported by National Natural Science Foundation Of China under Grants, 2006.

[9] Buyya R, Abramson D, Giddy J, Stockinger H, "Economic Models for Resource Management and Scheduling in Grid Computing". *Concurrency and Computation: Practice and Experience Journal (Special Issue on Grid Computing Environments)* 14 (13-15): 1507-1542, 2002

[10] Buyya R, Murshed M, Abramson D. A "Deadline and Budget constrained Cost-Time Optimization Algorithm for Scheduling Task Farming Applications on Global Grids[A]", *Proceeding of the 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '02)[C]*, 2002.

[11] Lijuan Xiao, Yanmin Zhu, Lionel M. Ni and Zhiwei Xu, "Incentive – Based Scheduling for Market – Like Computational Grids", *IEEE Transactions on Parallel and Distributed Systems*, vol 19, No 7, pp. 903-913, July 2008

[12] Zhang Jinqun, Ni Lina, Jiang Changjun. "A Heuristic Scheduling Strategy for Independent Tasks on Grid", *Proceedings of the Eighth International Conference on High-Performance Computing in Asia-Pacific Region (HPCASIA '05)*, 2005.